PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G11B 31/00, G06F 15/44

A1

(11) International Publication Number: **WO 97/15926**

(43) International Publication Date: 1 May 1997 (01.05.97)

(21) International Application Number: PCT/IB96/01056

(22) International Filing Date: 7 October 1996 (07.10.96)

(30) Priority Data:

115552	8 October 1995 (08.10.95)	IL
60/008,874	19 December 1995 (19.12.95)	US

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, HU, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, TJ, TM, TR, TT, UA, UG, US, UZ, VN, ARIPO patent (KE, LS, MW, SD, SZ, UG), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(71) Applicant *(for all designated States except US)*: FACE IMAGING LTD. [IL/IL]; 23 Hillel Street, Jerusalem 94581 (IL).

(72) Inventors; and

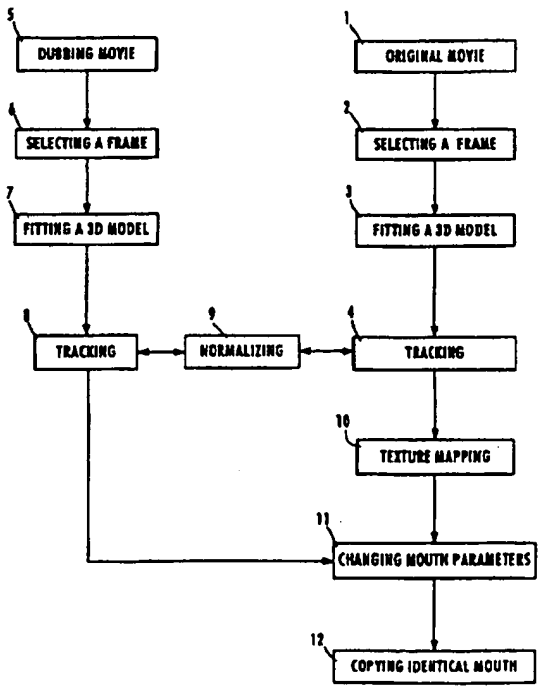
(75) Inventors/Applicants *(for US only)*: PELEG, Shmuel [IL/IL]; 45 Bar-Cochva Street, Jerusalem 97892 (IL). COHEN, Ran [IL/IL]; 6 Yekotiel Adam Street, Petach Tikva 49326 (IL). AVNIR, David [IL/IL]; 2 Novomeiski Street, Jerusalem 96908 (IL).

(74) Agent: NOAM, Meir; c/o Hauptman, Benjamin, J., Lowe, Price, Leblanc & Becker, Suite 300, 99 Canal Center Plaza, Alexandria, VA 22314 (US).

(54) Title: A METHOD FOR THE AUTOMATIC COMPUTERIZED AUDIO VISUAL DUBBING OF MOVIES

(57) Abstract

A method using computer software for automatic audio visual dubbing (5), using an efficient computerized automatic method for audio visual dubbing of movies by computerized image copying of the characteristic features of the lip movements of the dubber onto the mouth area of the original speaker. The invention uses a method of vicinity-searching, three-dimensional head modeling of the original speaker (3), and texture mapping (10) technique to produce new images which correspond to the dubbed sound track. The invention thus overcomes the well known disadvantage of the correlation problems between lip movement in an original movie and the sound track of the dubbed movie.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AM	Armenia	GB	United Kingdom	MW	Malawi
AT	Austria	GE	Georgia	MX	Mexico
AU	Australia	GN	Guinea	NE	Niger
BB	Barbados	GR	Greece	NL	Netherlands
BE	Belgium	HU	Hungary	NO	Norway
BF	Burkina Faso	IE	Ireland	NZ	New Zealand
BG	Bulgaria	IT	Italy	PL	Poland
BJ	Benin	JP	Japan	PT	Portugal
BR	Brazil	KE	Kenya	RO	Romania
BY	Belarus	KG	Kyrgystan	RU	Russian Federation
CA	Canada	KP	Democratic People's Republic of Korea	SD	Sudan
CF	Central African Republic			SE	Sweden
CG	Congo	KR	Republic of Korea	SG	Singapore
CH	Switzerland	KZ	Kazakhstan	SI	Slovenia
CI	Côte d'Ivoire	LI	Liechtenstein	SK	Slovakia
CM	Cameroon	LK	Sri Lanka	SN	Senegal
CN	China	LR	Liberia	SZ	Swaziland
CS	Czechoslovakia	LT	Lithuania	TD	Chad
CZ	Czech Republic	LU	Luxembourg	TG	Togo
DE	Germany	LV	Latvia	TJ	Tajikistan
DK	Denmark	MC	Monaco	TT	Trinidad and Tobago
EE	Estonia	MD	Republic of Moldova	UA	Ukraine
ES	Spain	MG	Madagascar	UG	Uganda
FI	Finland	ML	Mali	US	United States of America
FR	France	MN	Mongolia	UZ	Uzbekistan
GA	Gabon	MR	Mauritania	VN	Viet Nam

A METHOD FOR THE AUTOMATIC COMPUTERIZED AUDIO VISUAL DUBBING OF MOVIES

FIELD OF THE INVENTION

The present invention relates to a method for automatic audio visual dubbing. More specifically the said invention relates to an efficient computerized automatic method for audio visual dubbing of movies by computerized image copying of the characteristic features of the lips movements of the dubber onto the mouth area of the original speaker. The present invention uses a method of vicinity-searching, three-dimensional head modeling of the original speaker, and texture mapping techniques in order to produce the new images which correspond to the dubbed sound track.

The invention overcomes the well known disadvantage of the correlation problems between lips movement in the original movie and the sound track of the dubbed movie.

DEFINITIONS OF TERMS RELATED TO THE INVENTION

First are provided some definitions of important key words employed in this specification.

Actor (the original actor) - an actor, speaker, singer, animated character, animal, an object in a movie, or a subject in a still photograph.

Audio Visual Dubbing - Manipulating, in one or more frames, the mouth area of the actor so that its status will be similar as much as possible to that of the dubber in the reference frame.

Correlation Function - A function describing the similarity of two image regions. The higher the correlation, the better is the match.

Dubber - The person or persons, who speak/narrate/sing/interpret the target text. The dubber can be the same as the actor.

Dubbing - Replacing part or all of one or more of the original sound tracks of a movie, with its original text or sounds (including the case of the silent track of a still photograph), by another sound track containing the target text and/or sound.

Edge Detector - A known image processing technique used to extract boundaries between image regions which differ in intensity and/or color.

Face Parametrization - A method that numerically describes the structure, location, and expression of the face.

Head Model - A three-dimensional wire frame model of the face that is controlled by numerous parameters that describe the exact expression produced by the model (i.e. smile, mouth width, jaw opening, etc.).

Movie (the original movie) - Any motion picture (e.g. cinematic feature film, advertisement, video, animated cartoon, still video picture, etc.). A sequence of consecutive pictures (also called frames) photographed in succession by a camera or created by an animator. In the case of the movie being a still photograph, all of the consecutive pictures are identical to each other. When shown in rapid succession an illusion of natural motion is obtained, except for the case of still pictures. A sound-track is associated with most movies, which contains speech, music, and/or sounds, and which is synchronized with the pictures, and in particular where the speech is synchronized with the lip movements of the actors in the pictures. Movies are realized in several techniques. Common methods are: (a) recording on film, (b) recording in analog electronic form ("video"), (c) recording in digital electronic form, (d) recording on chips, magnetic tape, magnetic disks, or optical disks, and (e) read/write by magnetic and/or optical laser devices. Finally, in our context, an "original movie" is also an audio-visual movie altered by the present

invention, which serves as a base for further alterations.

Original Text - A text spoken or sung by the actor when the movie is being made, and which is recorded on its sound track. The text may be narrated in the background without showing the speaker, or by showing a still photograph of the speaker.

Pixel - Picture element. A digital picture is composed of an array of points, called pixels. Each pixel encodes the numerical values of the intensity and of the color at the corresponding picture point.

Reference Similarity Frame - A picture (being a frame in the original movie, a frame in any other movie, or a still photograph) in which the original actor has the desired features of the mouth-shape and head posture suitable for the audio visually dubbed movie.

Target Text - A new vocal text, to replace the original vocal text of the actor. The target text may also be that which is to be assigned to an actor who was silent in the original movie. The new text can be in another language, to which one refers as DUBBING. However, this invention relates also to replacement of text without changing the language, with the original actor or with a dubber in

that same language. The target text may have the same meaning as the original text, but may have also a modified, opposite, or completely different meaning. According to one of many applications of the present invention, the latter is employed for creation of new movies with the same actor, without his/her/its active participation. Also included is new vocal text used to replace the null vocal text attached to one or more still photographs.

Texture Mapping - A well known technique in computer graphics which maps texture onto a three-dimensional wire frame model.

Two-Dimensional Projection - The result of the rendering of the three-dimensional face model onto a two-dimensional device like a monitor, a screen, or photographic film.

BACKGROUND OF THE INVENTION

Movies are often played to an audience that is not familiar with the original language, and thus cannot understand the sound track of such movies. Two well known common approaches exist to solve this problem. In one approach sub-titles in typed text of the desried language are added to the pictures, and the viewers are expected

to hear the text in a foreign language and simultaneously to read its translation on the picture itself. Such reading distracts the viewers from the pictures and from the movie in general. Another approach is dubbing, where the original sound-track with the original text is being replaced by another sound-track with the desired language. In this case there is a disturbing mis-match between the sound-track and the movements of the mouth.

There have been some earlier attempts to overcome these disadvantages, none of which have been commercialized because of inherent principal difficulties which made the practical execution unrealistic. Thus, in U.S. Patent Number 4,600,281 a method is described which performs the measurements of the shape of the mouth manually by a ruler or with a cursor, and corrects the mouth shape by moving pixels within each frame. As will be seen in the description of the invention, the method according to the present invention is inherently different and much superior in the following points: In the present invention the tracking of the shape of the mouth is done automatically and not manually. In the present invention changing the shape of the mouth is done by using a three-dimensional head model, for example like those described by P.Ekman and W.V.Friesen, (Manual for the Facial Action Unit System, Consulting Psychologist Press, Palo Alto 1977). In the present invention the mouth area of the

actor is replaced using the mouth area of a reference similarity frame. In the present invention mouth status parameters of the dubber are substituted for mouth status parameters of the actor.

The U.S. Patent Number 4,260,229 relates to a method of graphically creating lip images. This U.S. patent is totally different from the present invention: In the U.S. Patent, speech sounds are analyzed and digitally encoded. In the present invention no sound analysis is done; nor is any required at all.

To make for better viewing of the audio visually dubbed movie, the present invention provides a computerized method wherein, in addition to replacing the sound track to the target text, the mouth movements of the actor are being automatically changed to match the target text. The new mouth movements are linguistly accurate and visually natural looking according to all of the observable parameters of the actor's face.

SUMMARY OF THE INVENTION

The present invention provides a method for automated computerized audio visual dubbing of movies, comprising of the following steps (see Fig. 1):

(a) selecting from the movie a frame having a picture, preferably frontal, of the actor's head and, if available, a frame with its side profile;

(b) marking on the face several significant feature points and measuring their locations in the frame;

(c) fitting a generic three-dimensional head model to the actor's two-dimensional head picture by adapting the data of the significant feature points, as measured in stage (b), to their location in the model;

(d) tracking of the said fitted three-dimensional head model parameters throughout the movie, from one frame to its successor, iteratively in an automated computerized way and creating a library of reference similarity frames.

(e) taking a movie of a dubber wherein the dubber speaks the target text;

(f) repeating stages (a), (b), (c), and (d) with the dubber;

(g) normalizing the dubber's minimum and maximum values of each parameter to the actor's minimum and maximum valuse of the same parameters;

(h) mapping, on a frame to frame basis, the two-dimensional actors face onto its three-dimensional head model by using a texture mapping technique, making use of reference similarity frames;

(i) changing the texture mapped three-dimensional model obtained in stage (h) by replacing, on a frame to frame basis, the original mouth parameters with the mouth parameters as computed in stage (d) for the dubber and obtaining the parametric description for the new picture, with identical values to the original, except that the actor's mouth status resembles the mouth status of the dubber;

(j) texture mapping the lips area of the same actor from a frame in the movie, with identical or very similar mouth status to the desired new mouth status, onto the lips area of the actor's head model for the current frame and then projecting the lips area from the actor's head model onto the current new frame. (This stage is optional, depending on the application.)

By using the three dimensional head model one can control the audio visual dubbing process even if the actor is moving his head. In most applications about 15 significant feature points on the face are used in the

tracking stage, such as eye corners, mouth corners, and the nostrills. Only those feature points which are visible to the viewer (using the information available to the model) are tracked.

In the present invention, audio visual dubbing is normally used in conjunction with the use of audio dubbing; but one may also use it in conjunction with an audio track where no equivalent track exists in the original movie.

The method according to the present invention is useful for the audio visual dubbing of motion pictures such as cinematic feature films, advertisments, video, and animated cartoon. Also the audio visual dubbing of still photographs, wherein all of the frames of the movie are the same, is made possible by the present invention. For instance, still photographs are used for this type of movie in T.V. news programs where the reporter's voice is heard while a still photograph of him/her is shown.

Thus, according to the present invention even speachless actors, infants, animals, and inanimate objects can be audio visually dubbed to speak in any language.

According to our invention, the animation process saves much of the labor associated with the animation of the

mouth area.

The present invention further provides a computer program (see Appendix 1) for operating the computerized audio visual dubbing.

The present invention further relates to the library of reference similarity frames as created in Step d (above).

DETAILED DESCRIPTION OF THE INVENTION

Given an original movie where an actor speaks the original text, a movie of a dubber is made where the dubber speaks the target text in either another language or the same language. The movie of the dubber is taken while the dubber performs a routine dubbing adaptation of the original into the target text.

This invention provides a method for the changing of the actor's facial motions in the original movie to create a new movie having the sound-track in the target text from the movie of the dubber, while the pictures are of the original movie, whereas the motion of the mouth of the actor is modified to correspond to the new sound-track.

For brevity considerations, the description of this invention uses pictures in electronic digital form

(composed of an array of pixels), but movies in any other form are treatable as well. In these cases, the movie is translated to a digital form by existing techniques, manipulated in the digital form, and returned back to any desired form by known techniques.

A facial expression can be described by an "action unit", for example the Facial Action Coding System (FACS) by Ekman and Friesen (Ekman et. al.). Action units (AU) stands for a small change in the facial expression which depends on a conscious activation of muscles (H. Li, P. Roivainen, R. Forchheimer, 3-D Motion in Model-Based Facial Image Coding, IEEE Transactions in PAMI, 15 (2), 545--555 (1993)). The AU information is expressed in parameter form. Using the AU parameters, many facial expressions can be controlled. Parameters like face location and size, aspect-ratios of face regions, location of specific face fetures and many more.

As explained above, one of the stages of this invention is a three-dimensional parameterisation of the face. An example for one such model is the model of Parke (Fredric I. Parke, Parameterized Models for Fatial Animation, IEEE computer Graphics and Applications, 12 (11), 61-68, (1982)) which consists of about 25 parameters. Face parameters can be roughly divided into three main classes: structure parameters, location

parameters, and expression parameters.

Structure parameters are fixed for every head and include distance ratios between the mouth and the eyes, the mouth and the chin, width of the model, jaw width, etc. The location parameters are, for example: three parameters for three-dimensional rotation in space and three parameters for three-dimensional translation (position in the real world). The expression parameters are, for instance: mouth width, smile (as an example, the parameter values here may be 0.0 for a very sad mouth and 1.0 for a very happy mouth), jaw opening, upper lip lifting, lower lip lowering, lip thickness, and so on.

Using a face model, the present invention is centered on a computer program (see Appendix 1), which automaticly re-shapes the lip movements of the actor according to the lip movements of the dubber by searching for the nearest reference similarity frames. This computer program (software), or similar, is an integral and an important part of the present invention. The process, according to the present invention, is divided generally into the TRACKING phase and the NEW-MOVIE generation phase, as follows:

I. TRACKING phase

Step 1: The first step is personalizing the generic

three-dimensional face model for both the actor and the dubber. In order to modify the generic face model to fit a specific face, some additional information is needed. The generic model has to be translated, scaled and stretched to fit the given actor's face, from its initial position and setting. This is done by manually pointing using a pointing device such as a mouth, a touch screen, etc., several facial feature points on the actor's face, e.g. eye corners, mouth corners, top and bottom of face. Typically a total of approximately 15 feature points are used, but this number may vary according to specifications. These feature points are marked on one (any) of the frames in the movie, in which the actor, preferably, faces the camera. The computer program then calculates automatically the exact model parameter's modifications needed for its two-dimensional projection to fit the actor face on the movie frame. In addition to using the facial feature points and in order to increase accuracy, the model is also adjusted to match the head edges, which are computed using an edge detector. If a side view of the actor is available it can be used to set several depth parameters, such as face depth and nose length. Otherwise, the face depth is scaled by some predetermined scale which is set experimentally.

Step 2: After the generic face model has been personalized to the desired actor, face features in

several key frames in the movie are marked. The number of such frames can vary from a single first frame to about 5% of all frames, depending on the difficulty of the segment fitting the model to the actor using the marked facial features in those key frames achieves a stabilization of the automatic tracking (described later), and these key frames assures stable and continuous tracking. Next, the program calibrates according to several examples of mouth shapes, later to be used for mouth tracking. Finally, the range of the mouth parameters (mimimum and maximum values) for the specific actor are estimated using all the values of the model parameters fitted to all key frames.

Step 3: The next stage is the automatic tracking of the actor's face throughout the entire movie: This is performed from one frame to its successive frame, using the face model, in two steps: first, the two-dimensional face of the actor is mapped onto the three-dimensional face model using a texture-mapping technique. The model can now be altered by changing its parameters only, creating new, synthetic images, which are otherwise very similar to the original movie frames: everything remains unchanged except for different face location, its orientation, and its expressions. By using a minimization algorithm, either analytic or numerical (such as steepest descent algorithm), the program now computes those

parameters which maximize the correlation function between the face area of the actor in the next frame and the synthesized projection of the texture-mapped face model. The steepest descent algorithm increases or decreases parameters in the direction that increases the correlation function. It can either work for each parameter separately (until it maximizes the correlation), or it can modify all the parameters at once.

Step 4: After the model is locked on the head of the actor in the next frame, the mouth has to be tracked. This is done by first, checking the parameters of all of the mouths in the key frames and in several previous frames already tracked. Then, the frame that gives the higher correlation is choosen as a first guess for the tracking. Next, the same minimization algorithm used to track the global head motion is used, until the correlation function has maximized. The parameters describing the face model in the tracked frame are written into a file for later use.

Step 5: Steps 3 and 4 are repeated until the entire movie is processed. For best results, instead of executing this process serially from the first frame to the last, the key frames can be used as initial points of tracking. Every two consecutive key frames are used to track from

each of them to the frames between them. That way, stabilasation of the tracking is preserved.

Step 6: The tracking described above is applied to the dubber movie as well.

II. NEW MOVIE generation phase

This phase combines the tracking results of both the original and the dubber's movies, in order to synthesize the new audio visually dubbed movie. This audio visually dubbed movie, as explained above, is mostly formed out of the original movie, except for the face of the actor in this audio visually dubbed movie. This face is a texture-mapped face on the three-dimensional face model, synthesized, as described above, to fit the lip, mouth, and cheek shapes of the dubber at that particular time. Thus, the parameters of the face model computed as described in phase I, is used to produce the new audio visually dubbed movie, in which for every frame in the original movie, the mouth parameters are modified to those of the dubber. The exact process is as follows:

Step 7: For every frame in the original movie, the face of the actor is texture-mapped on the appropriate face model using the parameters that were calculated in step 3 for the original movie. The mouth parameters of the

dubber as calculated in step 3 are used as follows for the new audio visually dubbed movie.

Step 8: Once the desired mouth-shape of the actor is known, the original movie is being searched in the neighborhood of the current frame (approximately 0.1 - 10 seconds forwards and backwards in time) for a mouth that is most similar in shape or parameters to the new desired mouth. This search for the reference similarity frame takes into account the mouth-shape already chosen for the previous frame in order to make the mouth motion smooth and continuous. From the several (5 - 10) best fit mouths, the mouth which is picked is from the frame that is closest in time to the previous picked mouth.

Step 9: The mouth chosen in step 8 is texture-mapped into the mouth model using its pre-computed parameters. The face model parameters are then changed to the desired mouth shape, producing a very realistic new frame, which replaces the old frame in the original movie. The user of the program can choose the desired mouth area to be texture-mapped in place - it can be either the inside of the mouth, the whole mouth including the lips or even a bigger area. This procedure creates a synthesized image, in which the face around the mouth, and in particular the lips, are re-shaped according to the sound track, while retaining the familiar face of the original actor. Step

8 can also be skipped, so that the inside of the mouth will be empty. This is useful for making a talking movie from still picture, where the inside information of the mouth is missing; because in the reference frame similarity dictionary of the actor there does not exist any near fit of lip shapes for the target. This black interior can also be filled with visual color/texture.

Step 10: Finally, the sound track from the dubbed movie (target text) replaces the original text and sound.

It is to be noted that an animator using the invented software tool, is free to modify, set, or fix any of the head or mouth parameters, in both the original or audio visually dubbed movie, and even pick a specific mouth to be textured-mapped in place, as described in step NM8, all of these at any of the above stages. The tracking program is highly interactive and user-friendly.

The involved software (see Appendix 1) of this invention is very versatile, and can be used in a very wide array of sound/text replacement applications many of which have been mentioned above. The following are examples of some applications of the present invention:

Advertising: For products sold world-wide, an original advertising commercial can be manipulated to produce the

same commercial in any desired language. This saves the need to produce a new video for every country or language that the product is aimed for.

Another possibility is to edit a movie, by altering existing scenes without having to re-shoot them again. If, for example, after the movie production was over, the director/editor wishes to alter a specific scene, or change one sentence of a specific actor.

The present invention refers not only to narrated text but also to songs, operas, and music, opening the possibility to change the language of musical video clips.

The production of an animated cartoon is assisted by drawing a line segment for the actor's mouth, drawing a small actor's picture dictionary containing representive reference similarity frames with completely drawn mouths, and then allowing these lip-line segments to be replaced with the coresponding lip shapes of the dubber as are to be found in the actor's picture dictionary.

In general, applications of a method for automatic audio visual dubbing of movie include: cinematic movies, cartoons, documentaries, advertizing, news, educational programs, court documentations, speaches, lectures,

historical documentation, hearing committees, home videos, sports events, entertainment events, operas, musicals, musical video-clips, simultaneous translation, and adding speach to sequences of either original or added still frames of the aforesaid.

Furthermore, using the library of the previously described reference similarity frames, the present invention allows one to create new movies altogether, and also to convert background narrative to audio visual speach, and to audio-visualize any written text.

The present invention will be further described by Figures 1-4. These figures are solely intended to illustrate the prefered embodyment of the invention and are not intended to limit the scope of the invention in any manner. Likewise the attached software (Appendix 1) represents an example of the implementation of the method disclosed in the present patent and is not intended to limit the scope of said method in any way.

Figure 1 illustrates a block diagram showing the method stages, and its contents is is detailed below.

Figures 2a and 2b illustrate an example of significant points on a generic frontal picture of a head (Figure 2a) and a generic side profile picture of a head (Figure 2b).

Figure 3 shows an example of a generic wire frame face model.

For illustrative purposes one can take the significant points shown in Figures 2, measure them on pictures of a real actor, and apply them to a generic wire frame face model (Figure 3). Fitting a three-dimensional head model to the actor's two-dimensional head picture by adapting the data of the significant points, as measured, results in an integration, as can be seen in Figures 4a and 4b.

Figure 4a is an example showing how a custom fitted wire frame model fits onto a frontal view of an actor's face.

Figure 4b is an example showing how a custom fitted wire frame model fits onto a profile view of an actor's face.

Figure 1 illustrates a block diagram showing the method stages:

In the original movie (1) a frame is selected (2) having a nearly frontal picture of the original actor's head and, if available, a frame is also selected having a picture with his side profile.

A three-dimensional head model is fitted to the actor's

two dimensional head picture(s). This model can be controlled by several parameters such as for the position of the head and the status of the mouth. This fitting stage (3) is done by adapting the data of the significant points, as measured in the selected frame, to the model.

The next step (4) is the automated computerized tracking of the fitted three-dimensional head model parameters throughout the movie, from one frame to the next. One partial or complete three-dimensional head model for each frame where the actor appears, is used. Any of the resulting frames can serve as a reference similarity frame for the lips replacement.

A movie of the dubber is taken (5). In this movie the dubber faces the camera in most of the frames. The dubber speaks the target text in this movie.

The same process as applied to the original actor's movie is applied to the dubber's movie: A frame form the dubber's movie is selected (6) having a frontal picture of the dubber's head, and if available, a frame with a picture of his side profile. A three dimensional head model is fitted to the dubber's two dimensional head picture (7) by adapting the data of the significant points, as measured in the selected frame, to the

dubber's model. An automated computerized tracking (8) of the said dubber's fitted three dimensional head model parameters is taken throughout the movie, from one frame to the next.

The next stage in this method is to normalize (9) the dubber's minimum and maximum parameters to the actor's minimum and maximum parameters.

In a frame to frame fashion, the original actor's two dimensional face is mapped (10) onto his three dimensional head model. This mapping stage is done by using a texture mapping technique with the reference similarity frames. The result of this stage is one mapped three dimensional partial head model of the original actor for each frame of the original actor in the original movie; wherein the model, corresponding to a given frame may be complete in the case when the original frame contains a frontal view of the actor's face.

In the next stage the textured three dimensional model frames obtained for the original actor are changed (11) by replacing, on a frame to frame basis, the original mouth parameters with mouth parameters as computed for the dubber in the corresponding frames; correspondance being determined by the desired sound track substitution (i.e. the dubbing). Thus is obtained the parametric

description for a new picture identical to the original, except that the actor's mouth status resembles the mouth status of the dubber; wherein the new picture coresponds to a frame in the new audio visually dubbed movie.

In order to overcome difficulties like for example those that arise when the dubber in (8) opens a closed mouth in (4), a frame or frames in the original movie are sought whose mouth status is similar to the desired new mouth status. These frames, termed reference similarity frames, are usually but not necessarily in a temporal proximity to the processed frame, and the lips from that frame are copied using texture mapping (12) into the lips area in the new frame. The search for a reference similarity frame is an essential component of the present invention. We therefore repeat its definition: a reference similarity frame is a picture (being a frame in the original movie, a frame in any other movie, or a still photograph) in which the original actor has the desired features of the mouth-shape and head posture suitable for the audio visually dubbed movie.

Alternatively, the reference similarity frame may be taken from a compiled picture library of the original actor or of other actors.

The process (12) is repeated all over again for each frame, until the entire movie is converted.

FACE IMAGING Ltd source code for program 'dub'

The following printout includes the source for the following modules:

dub.c	-	main program
general.c	-	general functions and utilities
gimg.h	-	interface for gimg.c - image I/O and filtering
mv.h	-	interface for mv.c - moving handling
list.h	-	interface for list.c - list data type handling
io.c	-	I/O functions
texture.c	-	texture mapping stuff

These modules comprises the program 'dub' which automatically fixes the lips of a person in a movie to another person's lip shapes, as described in the invention description.

The modules included are the main modules. Utility modules, such as handling images, movies, etc. are not included.

```
*****************************************************************************
 * Module: dub.c
 * By: Ranco
 * History:
 *  18.8.94: creation.
*****************************************************************************/
#include <event.h>
#include "x_inc.h"
#include "defs.h"
#include "image.h"
#include "general.h"
#include "mv.h"
#include "vars.h"
#include "dub.h"
#include "gimg.h"
#include "io.h"
#include "texture.h"
#include <math.h>
#include <dmedia/cl_cosmo.h>
#include <audiofile.h>

Movie movie,out_movie1, out_movie2,trans_movie;
XtAppContext ac;
Widget   tl, da;
Display *dpy;
GC       gc;
char output_movie_name[MAX_NAME],  source_movie_name[MAX_NAME],
    trans_movie_name[MAX_NAME],dubbed_movie_name[MAX_NAME],
    force_frames_filename[MAX_NAME],full_dubbed_movie_name[MAX_NAME],
    audio_file_name[MAX_NAME];
int first_frame, last_frame, translator_first_frame;
int sample = 1,frame_range = FRAME_AREA, jpeg_quality;
float mouth_scale;
int avg_params_flag = FALSE, avg_width_flag = TRUE, force_frames_flag = FALSE;
int full_flag = FALSE;
Parameter **left_source_array, **right_source_array;
```

```
Parameter **left_trans_array, **right_trans_array;
extern int texture_map;
Gimg curr_frame1;
int one_frame_mode, normalize_mouth, verbose_mode;
int which_params[] =
{JAW_ROTATION,MTH_Yscl,RSE_UPLIP,LWRLIP_SHIN,UPLIP_SHIN,LWRLIP_FTUCK
,MTH_INTERP};
int ranges[] = {1,2,1,1,1,0,1};

AFfilehandle audio1 = AF_NULL_FILEHANDLE, audio2 = AF_NULL_FILEHANDLE;
int play_flag;
/*
 * GLX configuration parameter:
 *      Double buffering
 *      color index (default so unspecified)
 *      nothing else special
 */
static GLXconfig glxConfig [] = {
  { GLX_NORMAL, GLX_DOUBLE, FALSE },
  { GLX_NORMAL, GLX_ZSIZE, 7 },
  { GLX_NORMAL, GLX_RGB, TRUE },
  { 0, 0, 0 }
};

/* texture properties for the whole face */
float texprops1[] = {
  TX_MINFILTER, TX_POINT,
  TX_MAGFILTER, TX_POINT,
  TX_WRAP, TX_CLAMP, TX_NULL};
/* texture properties for the lips  */
float texprops2[] = {
  TX_MINFILTER, TX_BILINEAR,
  TX_MAGFILTER, TX_BILINEAR,
  TX_WRAP, TX_CLAMP, TX_NULL};

/* Texture environment for the face and the lips */
```

```
float tevprops1[] = {TV_DECAL, TV_NULL};

/*****************************************************************************/
void Refresh(int d1, int d2)
{
  if (RedrawNeeded) {
    recompute_face();
    DrawScene();
    RedrawNeeded = FALSE;
  }
}

/*****************************************************************************/
void cancelCB(Widget w, XtPointer xtp, GlxDrawCallbackStruct *call_data)
{
  exit(0);
}

/*****************************************************************************/
void da_initCB(Widget w, XtPointer xtp, GlxDrawCallbackStruct *call_data)
{
  zbuffer(TRUE);
  winconstraints();
  reshapeviewport();
  subpixel(TRUE);

  mmode(MVIEWING);
  ortho(-xsize/2.0+.5,xsize/2.0+.5,-ysize/2.0+.5,ysize/2.0+.5,-500.0,2000.0);

  zfar = getgdesc(GD_ZMAX);
  lsetdepth(0, zfar);
  zclear();
  cpack(bgc);
  clear();
  concave(FALSE);
```

```
  mmode(MPROJECTION);
  getmatrix(mp);
  mmode(MVIEWING);
  frontbuffer(TRUE);
  RedrawNeeded = TRUE;
}

/*****************************************************************************/
void da_resetCB(Widget w, XtPointer xtp, GlxDrawCallbackStruct *call_data)
{
  RedrawNeeded = TRUE;
}

/*****************************************************************************/
void UIInit(int *argc, char **argv)
{
  Widget cancel, form, rc,rc1, go, sep, label;
  int i;
  XColor color, uu;
  Colormap cmap;
  Pixel bg_color, top_shadow, bottom_shadow, fg, select_color;
  XmString frame_str = XmStringCreateSimple("Frame: "),
    label_str = XmStringCreateSimple("                    D E M O "),
    wf_str = XmStringCreateSimple(" 1/1 "),
    go_str = XmStringCreateSimple(" Go!"),
    cancel_str = XmStringCreateSimple("Quit");

  tl = XtAppInitialize (&ac, "Dub", NULL, 0, argc, argv, NULL, NULL, 0);
  dpy = XtDisplay (tl);
  form = XtVaCreateManagedWidget("form", xmFormWidgetClass, tl,
                        XmNnoResize, TRUE,
                        NULL);

  da = XtVaCreateManagedWidget ("da", glxMDrawWidgetClass, form,
                        GlxNglxConfig, glxConfig,
                        XmNwidth, xsize,
```

```
                    XmNheight, ysize,
                    XmNbottomAttachment, XmATTACH_FORM,
                    XmNtopOffset, 2,
                    XmNbottomOffset, 2,
                    XmNbackground, WHITE,
                    NULL);
  XtAddCallback(da, GlxNresizeCallback, (XtCallbackProc)da_resetCB, NULL);
  XtAddCallback(da, GlxNexposeCallback, (XtCallbackProc)da_resetCB, NULL);
  XtAddCallback(da, GlxNginitCallback, (XtCallbackProc)da_initCB, NULL);
  XtRealizeWidget (tl);
}

/*****************************************************************************/
void Init(int argc, char **argv)
{
  char vbuff[50];
  long zbuf = getgdesc(GD_BITS_NORM_ZBUFFER);
  long rgb_sbits[3], rgb_dbits[3];
  int i, ch, source_movie_name_flag = 0, trans_movie_name_flag = 0,last_frame_flag=0;
  int n_frames,prm;
  char model_filename[MAX_NAME];
  char temp[MAX_NAME], params[MAX_NAME];


  xsize = getgdesc(GD_XPMAX);

  if (argc < 5)
    ProgUsage(argv[0]);

  /* See if we're on a GT. */
  gversion( vbuff );
  rgb_sbits[0] = getgdesc(GD_BITS_NORM_SNG_RED);
  rgb_sbits[1] = getgdesc(GD_BITS_NORM_SNG_GREEN);
  rgb_sbits[2] = getgdesc(GD_BITS_NORM_SNG_BLUE);
  rgb_dbits[0] = getgdesc(GD_BITS_NORM_DBL_RED);
  rgb_dbits[1] = getgdesc(GD_BITS_NORM_DBL_GREEN);
```

```
rgb_dbits[2] = getgdesc(GD_BITS_NORM_DBL_BLUE);
ok_for_shading= (
            (rgb_sbits[0]>0) &&
            (rgb_sbits[1]>0) &&
            (rgb_sbits[2]>0));

ok_for_double_buffering= (
                  (rgb_dbits[0]>=OK_FOR_DOUBLE_BUFFERING)  &&
                  (rgb_dbits[1]>=OK_FOR_DOUBLE_BUFFERING)  &&
                  (rgb_dbits[2]>=OK_FOR_DOUBLE_BUFFERING));

if(debug) {
  fprintf(stdout,"Machine is %s at res %ld\n",vbuff,xsize);
  fprintf(stdout,"zbuffer bits is %ld\n",zbuf);
  fprintf(stdout,"rgb single is %ld,%ld,%ld\n",rgb_sbits[0],rgb_sbits[1],rgb_sbits[2]);
  fprintf(stdout,"rgb double is %ld,%ld,%ld\n",rgb_dbits[0],rgb_dbits[1],rgb_dbits[2]);
}

/* initialize some globals */
prefix = "DATA/face_model";
texture_map = FALSE;
param_face = TRUE;
eye_position_not_set = TRUE;
show_model = TRUE;
lips_only = FALSE;
put_image = TRUE;
which_image = FRONT_IMG;
show_teeth = FALSE;
show_coord = TRUE;
wire = TRUE;
one_frame_mode = FALSE;
mouth_area = TEX_NONE;
normalize_mouth = TRUE;
verbose_mode = FALSE;
current_window = MAIN;
curr_saved_image = 1;
```

```
RedrawNeeded = RecalcNeeded = TRUE;
mouth_scale = 1.0;
first_frame = translator_first_frame = DEF_FIRST_FRAME;
strcpy(output_movie_name, DEF_OUTPUT_MOVIE_NAME);
params[0] = 0;
jpeg_quality = 93;
play_flag = FALSE;
bgc = 0x00202020;
i = 1;
tex_mode = DUB_MODE;

while(--argc && (ch = argv[i++][1])) {
 switch (ch) {
  case 's':   /* Source movie */
   strcpy(source_movie_name, argv[i++]);
   source_movie_name_flag++;
   break;
  case 'o':   /* New movie */
   strcpy(output_movie_name, argv[i++]);
   break;
  case 't':
   strcpy(trans_movie_name, argv[i++]);
   trans_movie_name_flag++;
   break;
  case 'u':
   strcpy(force_frames_filename, argv[i++]);
   force_frames_flag++;
   break;
  case 'f':
   first_frame = atoi(argv[i++]);
   break;
  case 'l':
   last_frame = atoi(argv[i++]);
   last_frame_flag++;
   break;
  case 'q':
```

```
    jpeg_quality = atoi(argv[i++]);
    sprintf(temp,"-q_%d",jpeg_quality);
    strcat(params,temp);
    break;
  case 'm':
    if (strncmp(argv[i-1],"-mc",strlen("-mc")) == 0) {
      sscanf(argv[i++],"%lx",&bgc);
      printf("Mouth color = %x\n",bgc);
    }
    else {
      mouth_scale = atof(argv[i++]);
      sprintf(temp,"-m_%.2f",mouth_scale);
      strcat(params,temp);
    }
    break;
  case 'F':
    translator_first_frame = atoi(argv[i++]);
    break;
  case 'r':
    frame_range = atoi(argv[i++]);
    sprintf(temp,"-r_%d",frame_range);
    strcat(params,temp);
    break;
  case 'w':
    strcat(params,"-w");
    wire = TRUE;
    put_image = FALSE;
    argc++;
    break;
  case 'a':
    if (argv[i-1][2] == 'p') avg_params_flag = TRUE;
    else if (argv[i-1][2] == 'w') avg_width_flag = TRUE;
    strcat(params,argv[i-1]);
    argc++;
    break;
  case 'b':
```

```
        full_flag = TRUE;
        argc++;
        break;
      case 'n':
        normalize_mouth = FALSE;
        argc++;
        break;
      case 'I':   /* use INside of mouth */
        if (argv[i-1][2] == 'N') mouth_area = TEX_IN;
        strcat(params,"-IN");
        argc++;
        break;
      case 'E':   /* use EXTERNAL mouth */
        if (!strncmp(argv[i-1]+2, "XT",2)) mouth_area = TEX_EXT;
        strcat(params,"-EXT");
        argc++;
        break;
      case 'O':   /* use OUTside of mouth */
        if (argv[i-1][2] == 'U') mouth_area = TEX_OUT;
        strcat(params,"-OUT");
        argc++;
        break;
      case 'v':
        verbose_mode = TRUE;
        argc++;
        break;
      default:
        fprintf(stderr, "%c is an illegal flag!\n", (char)ch);
        argc++;
        break;
      }
      argc--;
      }

  if (!source_movie_name_flag || !trans_movie_name) ProgUsage(argv[0]);
```

```
make_face();    /* read data, topology and parameters. */

if (!put_image) {
  fgc=0xFF000000;          /* set background color. */
  bgc=0xFFFFFFFF;          /* set foreground color. */
}
object_is_visible[FLESH]    = TRUE;
object_is_visible[LIPS]     = TRUE;
object_is_visible[EYELASH]  = TRUE;
object_is_visible[TEETH]    = FALSE;
object_is_visible[PUPIL]    = FALSE;
object_is_visible[IRIS]     = FALSE;
object_is_visible[FRINGE]   = FALSE;
object_is_visible[EYEWHITE] = FALSE;
setPolarAzimuth(0.0);
setPolarInc(90.0);
setPolarTwist(0.0);
saved_front_sc = saved_side_sc = sc;

sprintf(model_filename,"%s.model",source_movie_name);
restoreModel(model_filename);
sprintf(temp, "%s.%d.prm", source_movie_name, first_frame);
PRINTF1("Restoring params from %s\n", temp);
restoreParameters(temp,left_parameter,right_parameter,FALSE);

mth_color = 0l;

/* Create output movies and open input movies */
if ((movie = MOVIE_open(source_movie_name)) == NULL) {
  exit(0);
}
xsize = MOVIE_frameWidth(movie);
ysize = MOVIE_frameHeight(movie);
printf("Source frame size is (%dx%d)\n",xsize,ysize);

if ((trans_movie = MOVIE_open(trans_movie_name)) == NULL) {
```

```
  exit(0);
 }
 sprintf(audio_file_name,"%s.a",trans_movie_name);
 if (full_flag) {
  sprintf(full_dubbed_movie_name,"%s%s_FULLSIZE",output_movie_name,params);
  printf("Creating movie: %s.mv\n",full_dubbed_movie_name);
  out_movie1  = MOVIE_create(full_dubbed_movie_name,xsize,ysize,
                        MOVIE_frameRate(trans_movie),
                        MOVIE_audioRate(trans_movie),
                        DM_TOP_TO_BOTTOM,
                        DM_IMAGE_JPEG,
                        DM_IMAGE_INTERLACED_EVEN,
                        jpeg_quality);
  audio1 = MOVIE_createAudioTrack(audio_file_name,out_movie1);
 }
 else {
  sprintf(dubbed_movie_name,"%s%s",output_movie_name,params);
  printf("Creating movie: %s.mv\n",dubbed_movie_name);
  out_movie2 = MOVIE_create(dubbed_movie_name,xsize/2,ysize/2,
                        MOVIE_frameRate(trans_movie),
                        MOVIE_audioRate(trans_movie),
                        DM_BOTTOM_TO_TOP,
                        DM_IMAGE_MVC1,
                        DM_IMAGE_NONINTERLACED,
                        jpeg_quality);
  audio2 = MOVIE_createAudioTrack(audio_file_name,out_movie2);
 }
 if (!last_frame_flag) last_frame = MOVIE_numOfFrames(movie)-1;
 if (MOVIE_numOfFrames(movie)  == 1) one_frame_mode = TRUE;

 curr_frame1 = GimgCreate (xsize,ysize,GRGBA);

 n_frames = last_frame + 1;

 MALLOC(left_source_array,n_frames,Parameter_ptr);
 MALLOC(right_source_array,n_frames,Parameter_ptr);
```

```
  restoreAllParameters(source_movie_name,n_frames,
                       left_source_array,right_source_array);
  MALLOC(left_trans_array,n_frames,Parameter_ptr);
  MALLOC(right_trans_array,n_frames,Parameter_ptr);
  restoreAllParameters(trans_movie_name,n_frames,
                       left_trans_array,right_trans_array);

  if (verbose_mode) {
    for (i=0; i < XtNumber(which_params); i++) {
      prm = which_params[i];
      printf("%s\t",left_parameter[prm].box_label);
    }
  }
}

/*****************************************************************************/
float calcParamDist (int frame_no)
{
  float distance=0, min_dist = SOME_BIG_NUMBER, t[5], range;
  Parameter *p_l1, *p_l2, *p_r1, *p_r2;
  int k;

  p_l1 = left_source_array[frame_no];
  p_r1 = right_source_array[frame_no];
  p_l2 = left_parameter;
  p_r2 = right_parameter;

  range = MAX_VAL(p_l1[JAW_ROTATION])-MIN_VAL(p_l1[JAW_ROTATION]);
  t[0] = (VAL(p_l1[JAW_ROTATION])-VAL(p_l2[JAW_ROTATION]))/range;

  range = MAX_VAL(p_l1[MTH_Yscl])-MIN_VAL(p_l1[MTH_Yscl]);
  t[1] = (VAL(p_l1[MTH_Yscl])-VAL(p_l2[MTH_Yscl]))/range;
  t[2] = (VAL(p_r1[MTH_Yscl])-VAL(p_r2[MTH_Yscl]))/range;

  range = MAX_VAL(p_l1[RSE_UPLIP])-MIN_VAL(p_l1[RSE_UPLIP]);
  t[3] = (VAL(p_l1[RSE_UPLIP])-VAL(p_l2[RSE_UPLIP]))/range;
```

```
  range = MAX_VAL(p_l1[LWRLIP_FTUCK])-MIN_VAL(p_l1[LWRLIP_FTUCK]);
  t[4] = (VAL(p_l1[LWRLIP_FTUCK])-VAL(p_l2[LWRLIP_FTUCK]))/range;

  for (k=0; k<5; k++) distance += fabs(t[k]);

  return (distance);
}

/*****************************************************************************/
int dist_comp (const void *d1, const void *d2)
{
  float diff = ((dist_st *)d1)->distance - ((dist_st *)d2)->distance;
  if (diff < 0.0) return -1;
  else if (diff > 0.0) return 1;
  return 0;
}

/*****************************************************************************/
int angleTooFar (int a)
{
  Parameter *ls = left_source_array[a], *lp = left_parameter;
  float twist1 = VAL(ls[POLAR_TWIST]), twist2 = VAL(lp[POLAR_TWIST]);

  if (twist1 >= 180.0) twist1 -= 360.0;
  if (twist2 >= 180.0) twist2 -= 360.0;
  if ((fabs(VAL(ls[POLAR_AZIM])-VAL(lp[POLAR_AZIM]))  > 10.0) ||
     (fabs(VAL(ls[POLAR_INC])-VAL(lp[POLAR_INC]))  > 10.0) ||
     (fabs(twist1-twist2) > 10.0)) {
   return (TRUE);
  }
  return (FALSE);
}

/*****************************************************************************/
int scaleTooFar (int a)
{
```

```
 Parameter *ls = left_source_array[a], *lp = left_parameter;
 float ratio = VAL(ls[SCALE])/VAL(lp[SCALE]);

 if (ratio < 0.9 || ratio > 1.1) return (TRUE);
 return (FALSE);
}

/*****************************************************************************/
int loadMouth (int frame_no, int last_chosen)
{
 int i, j, k, best_fit = SOME_BIG_NUMBER,  width,height,index = SOME_BIG_NUMBER,
    closest, d, found1, bigger_mouth;
 int from  = MAX(first_frame,frame_no-frame_range),
    to    = MIN(last_frame,frame_no+frame_range);
 float distance, min_dist = SOME_BIG_NUMBER;
 char temp[MAX_NAME];
 Gimg sub_img;
 dist_st *distances;

 for (i=from; i < to; i++) {
    distance = calcParamDist(i);
    distance /= fsin(RAD(1.5*(i-last_chosen)+90.0));
    if (distance-min_dist<-0.000001) {
      min_dist = distance;
      best_fit = i;

 printf("Frame#%d: best fit is frame#%d (%f)\n", frame_no,best_fit,min_dist);


 MOVIE_getFrame (movie, best_fit,curr_frame1);

 sprintf(temp,"%s.%d.tex",source_movie_name,  best_fit);
 restoreTexCoord(temp,left_mth_pts,right_mth_pts,
            left_mth_tex,right_mth_tex); /* Also sets model region vars */
 recompute_face();
 width = (int)(right_model-left_model);
```

```
  height = (int)(top_model-bottom_model);

  sub_img = GimgCreateSubCopy(curr_frame1,(int)left_model,
                        (int)bottom_model,width,height);
  tevdef(1, 0, tevprops1);

  GimgSetAlpha(sub_img,255l);
  texdef2d(2, 4, width,height, (u_long *)ImgRast(sub_img), 7, texprops1);
  GimgSetAlpha(sub_img,255l);
  texdef2d(3, 4, width,height, (u_long *)ImgRast(sub_img), 7, texprops1);
  GimgDel(sub_img);
  return (best_fit);
}


/*****************************************************************************/
int insertMouth (int frame_no, int best_fit)
{
  int i, j, k,width,height;
  char temp[MAX_NAME];
  Gimg sub_img;

  printf("Frame#%d: mouth taken from frame#%d\n", frame_no, best_fit);
  loadActorLipsParameters(best_fit);
  DrawScene();
  MOVIE_getFrame (movie, best_fit,curr_frame1);
  sprintf(temp,"%s.%d.tex",source_movie_name, best_fit);
  restoreTexCoord(temp,left_mth_pts,right_mth_pts,
              left_mth_tex,right_mth_tex); /* Also sets model region vars */
  recompute_face();
  width = (int)(right_model-left_model);
  height = (int)(top_model-bottom_model);

  tevdef(1, 0, tevprops1);

  sub_img =GimgCreateSubCopy(curr_frame1,(int)left_model,
```

```
                    (int)bottom_model,width,height);
  GimgSetAlpha(sub_img,255l);
  texdef2d(2, 4, width,height, (u_long *)ImgRast(sub_img), 7, texprops1);
  GimgSetAlpha(sub_img,255l);
  texdef2d(3, 4, width,height, (u_long *)ImgRast(sub_img), 7, texprops1);
  GimgDel(sub_img);
  return (best_fit);
}

/*****************************************************************************/
void loadParameters(Parameter *left_array, Parameter *right_array)
{
  int i;
  for (i=0; i<MAX_PARAMETERS; i++) {
    left_parameter[i]  = left_array[i];
    right_parameter[i] = right_array[i];
  }
}

/*****************************************************************************/
float normalizedParam(Parameter **trans_params,
                      Parameter *source_params,
                      int frame_no,
                      int prm)
{
  float res, v, max_t, min_t, min_s, max_s;

  min_t = MIN_VAL(trans_params[0][prm]);
  max_t = MAX_VAL(trans_params[0][prm]);
  min_s = MIN_VAL(source_params[prm])*mouth_scale;
  max_s = MAX_VAL(source_params[prm])*mouth_scale;

  v = (VAL(trans_params[frame_no][prm])-min_t)/(max_t-min_t);

  res = v * (max_s-min_s) + min_s;
```

```
 return (res);
}

/*****************************************************************************/
void smoothLipsParameters(int frame_no, int prm, int range, float *l, float *r,
                          int normalize_flag)
{
 float ll = 0.0, rl = 0.0, min_l = SOME_BIG_NUMBER,
       min_r = SOME_BIG_NUMBER, max_l = -SOME_BIG_NUMBER, max_r =
-SOME_BIG_NUMBER;
 float cur_l, cur_r;
 int i, from  = MAX(first_frame,frame_no-range),
        to    = MIN(last_frame,frame_no+range);

 if (normalize_flag) {
  cur_l = normalizedParam(left_trans_array,left_source_array[0],frame_no,prm);
  cur_r = normalizedParam(right_trans_array,right_source_array[0],frame_no,prm);
  for (i=from-frame_no; i <= to-frame_no; i++) {
   ll += normalizedParam(left_trans_array,left_source_array[0],frame_no+i,prm);
   rl += normalizedParam(right_trans_array,right_source_array[0],frame_no+i,prm);
   if (ll < min_l) min_l = ll;
   if (ll > max_l) max_l = ll;
   if (rl < min_r) min_r = rl;
   if (rl > max_r) max_r = rl;
  }
 }
 else {
  if (prm == MTH_Yscl) {
   cur_l = VAL(left_trans_array[frame_no][prm])*mouth_scale;
   cur_r = VAL(right_trans_array[frame_no][prm])*mouth_scale;
  }
  else {
   cur_l = VAL(left_trans_array[frame_no][prm]);
   cur_r = VAL(right_trans_array[frame_no][prm]);
  }
  for (i=from-frame_no; i <= to-frame_no; i++) {
```

```
    if (prm == MTH_Yscl) {
      l1 += VAL(left_trans_array[frame_no+i][prm])*mouth_scale;
      r1 += VAL(right_trans_array[frame_no+i][prm])*mouth_scale;
    }
    else {
      l1 += VAL(left_trans_array[frame_no+i][prm]);
      r1 += VAL(right_trans_array[frame_no+i][prm]);
    }
    if (l1 < min_l) min_l = l1;
    if (l1 > max_l) max_l = l1;
    if (r1 < min_r) min_r = r1;
    if (r1 > max_r) max_r = r1;
   }
  }
  *l = l1 / (to-from+1);
  *r = r1 / (to-from+1);
}


/*****************************************************************************/
void loadTranslatorLipsParameters(int  frame_no, int normalize_params)
{
  float l1,l2,l3,r1,r2,r3;
  int i, prm;
  int from  = MAX(first_frame,frame_no-frame_range),
      to    = MIN(last_frame,frame_no+frame_range);

  for (i=0; i < XtNumber(which_params); i++) {
    prm = which_params[i];

    if (avg_params_flag)
      smoothLipsParameters(frame_no,prm,  ranges[i],&l2, &r2, normalize_params);
    else
      smoothLipsParameters(frame_no,prm,  0,        &l2, &r2, normalize_params);

    if (prm == MTH_Yscl || prm == MTH_INTERP) {
```

```
      if (avg_width_flag)
        VAL(left_parameter[prm]) = VAL(right_parameter[prm]) = (l2+r2)/2.0;
    }
    else {
      VAL(left_parameter[prm])  = l2;
      VAL(right_parameter[prm]) = r2;
    }
    PRINTF2("changed to:(%f,%f)\n",VAL(left_parameter[prm]),VAL(right_parameter[prm]));

    if (verbose_mode) printf("(%.4f,%.4f)\t",l2,r2);
  }

  if (verbose_mode) printf("\n");
  recompute_face();
}

/*****************************************************************************/
void loadActorLipsParameters(int  frame_no)
{
  int i, prm;
  float l1,l2,l3,r1,r2,r3;

  for (i=0; i < XtNumber(which_params); i++) {
    prm = which_params[i];
    VAL(left_parameter[prm]) = VAL(left_source_array[frame_no][prm]);
    VAL(right_parameter[prm]) = VAL(right_source_array[frame_no][prm]);;
  }
  recompute_face();
}

/*****************************************************************************/
void Go(void)
{
  long nof;                    /* Num of frames */
  char temp[MAX_NAME], line[80];
  int i, j, last, orig_frame_no,new_frame_no;
```

```
Gimg small_img, grab_frame;
time_t tm;
float p16[2],p288[2], distance,distance1,sb;
Matrix M;
FILE *force_frames;
int compressed_image_size, *compressed_image;

if (force_frames_flag) {
 if ((force_frames = fopen(force_frames_filename, "r")) == NULL)
  force_frames_flag = FALSE;
 else {
  fgets (line, 80, force_frames);
  sscanf(line, "%d\t%d", &orig_frame_no, &new_frame_no);
 }
}

tm = time(0);
printf("Start dubbing %d frames at %s",last_frame-first_frame, ctime(&tm));

readsource(SRC_FRONT);
grab_frame = GimgCreate(xsize,ysize,GRGBA);

MOVIE_setCurrentFrameNo (movie, first_frame);
if (one_frame_mode) {
 textureMapping(TEX_ON);
 DrawScene();
}

cpack(bgc);
for (i=first_frame, j=translator_first_frame; i <= last_frame; i++,j++) {
 tex_mode = TRACK_MODE;
 sprintf(temp, "%s.%d.prm", source_movie_name, i);
 if (!one_frame_mode) loadParameters(left_source_array[i],right_source_array[i]);
 /*   else loadParameters(left_source_array[0],right_source_array[0]);*/
 else {
  VAL(left_parameter[BEARD_SCL])  = VAL(right_parameter[BEARD_SCL])  = 0;
```

```
  recompute_face();
 }

 VAL(left_parameter[BEARD_SCL])  = VAL(right_parameter[BEARD_SCL])  = 0;

 recompute_face();
 DrawScene();
 prepareMatrix(M);
 if (put_image && !one_frame_mode) {
  textureMapping(TEX_ON);
  DrawScene();

 }
 loadTranslatorLipsParameters(i,normalize_mouth);
 DrawScene();
 calcProjection(left_trans_pts,right_trans_pts);
 if (mouth_area >= TEX_IN) {
  if (force_frames_flag && (i == orig_frame_no)) {
    last = insertMouth(orig_frame_no,new_frame_no);
    fgets (line, 80, force_frames);
    sscanf(line, "%d\t%d", &orig_frame_no, &new_frame_no);
  }
  else {
    if (i == first_frame)
      last = loadMouth(i,i);
    else
      last = loadMouth(i,last);
  }
 }
 else
  printf("Frame#%d\n", i);

 tex_mode = DUB_MODE;
 recompute_face();
 prepareMatrix(M);
 DrawScene();
```

```
    lrectread(0,0,xsize-1,ysize-1,(u_long *)ImgRast(grab_frame));
    MOVIE_addCurrentFrame(out_movie1,grab_frame,0);
    MOVIE_addAudioFrame(audio1,out_movie1);
    }
    else {
      small_img = GimgResize(grab_frame,xsize/2,ysize/2,2);
      MOVIE_addCurrentFrame(out_movie2,small_img,0);
      MOVIE_addAudioFrame(audio2,out_movie2);
      GimgFree(small_img);
    }
    if (put_image && !one_frame_mode) {
      MOVIE_nextFrame(movie);
      textureMapping(TEX_OFF);
    }
  }

  if (full_flag) {
    MOVIE_close(out_movie1);
    out_movie1  = MOVIE_open(full_dubbed_movie_name);
  }
  else {
    MOVIE_close(out_movie2);
    out_movie2  = MOVIE_open(dubbed_movie_name);
  }
  tm = time(0);
  AFclosefile(audio1);
  AFclosefile(audio2);
  printf("Finished at %s", ctime(&tm));
}

/*****************************************************************************/
void DrawScene(void)
{
  zclear();
  update_window_display();
}
```

```
/*************************************************************************/
void ProgUsage(char *filename)
{
  fprintf(stderr,"Dub: fix lips in a dubbed movie.\n");
  fprintf(stderr,"Usage:\n");
  fprintf(stderr,"  %s [options]\n", filename);
  fprintf(stderr,"\n Options:\n");
  fprintf(stderr,"  -s <movie file>\tSource: movie to be dubbed\n");
  fprintf(stderr,"  -o <movie file>\tOutput movie name (default=\"movie\")\n");
  fprintf(stderr,"  -t <movie file>\tTranslator's movie\n");
  fprintf(stderr,"  -u <filename>\t\tForce usage of info in filename to plant mouth\n");
  fprintf(stderr,"  -f <integer>\t\tFirst frame num, starting at 0 (default)\n");
  fprintf(stderr,"  -l <integer>\t\tLast frame num (default=0)\n");
  fprintf(stderr,"  -F <integer>\t\tTranslator's First frame num (default=0)\n");
  fprintf(stderr,"  -q <integer>\t\tJpeg quality (default=93)\n");
  fprintf(stderr,"  -m <integer>\t\tMouth scale factor (optional)\n");
  fprintf(stderr,"  -r <integer>\t\tRange of frame to search mouth (defalut=25)\n");
  fprintf(stderr,"  -w \t\t\tCreate only Wire frame animation (optional)\n");
  fprintf(stderr,"  -ap \t\t\tAverage params (optional)\n");
  fprintf(stderr,"  -aw \t\t\tAverage mouth width (optional)\n");
  fprintf(stderr,"  -b \t\t\tCreate a full size movie (optional)\n");
  fprintf(stderr,"  -n \t\t\tDon't Normalize mouth\n");
  fprintf(stderr,"  -IN \t\t\tUse inside of mouth (default)\n");
  fprintf(stderr,"  -OUT \t\t\tUse outside of mouth\n");
  fprintf(stderr,"  -EXT \t\t\tUse bigger area around mouth\n");
  fprintf(stderr,"  -mc \t\t\tInside mouth color\n");
  fprintf(stderr,"  -v \t\t\tVerbose mode\n");

  exit(0);
}

/*************************************************************************/
main(int argc, char **argv)
{
  XEvent e;
  char temp[200];
```

```
  Init(argc, argv);
  UIInit(&argc, argv);
  draw_poly = 1;
  MOVIE_bind(movie, da);
  MOVIE_setCurrentFrameNo(movie,first_frame);
  draw_face();
  RedrawNeeded = TRUE;
  while (XtAppPending(ac)){
    XtAppNextEvent (ac, &e);
    XtDispatchEvent (&e);
  }
  Go();
  printf("Done!\n");
}
/*****************************************************************************
 * Module: general.c
 * By: Ranco
 * History:
 *  18.8.94: creation.
*****************************************************************************/
#include "x_inc.h"
#include "mv.h"
#include "general.h"

/*****************************************************************************
 Set the proper GL matrices: The transformation that a point p experience
   is as follows:

   p * (S * Rx(-90)*Ry(-90)*Rx(90-x)*Ry(-y)Rz)  * TRANS * Tnew = (U', V')

*****************************************************************************/
void setMatrices(void)
{
  loadmatrix(idmat);
  translate (left_parameter[HEAD_Yoff].value,
             left_parameter[HEAD_Zoff].value,
```

```
            left_parameter[HEAD_Xoff].value);
  rotateAndScale();
}

/*****************************************************************************/
void rotateAndScale(void)
{
  /* Global transformation:    */
  /*  multmatrix(global_trans);*/

  rotate((int)(left_parameter[POLAR_TWIST].value*10),'z');
  rotate(-(int)(left_parameter[POLAR_AZIM].value*10),'y');
  rotate(900-(int)(left_parameter[POLAR_INC].value*10),'x');
  rotate(-900,'y');
  rotate(-900,'x');
  if (which_image == SIDE_IMG)
    scale(saved_side_sc,saved_side_sc,saved_side_sc);
  else
    scale(sc,sc,sc);
}

/*****************************************************************************/
void get_composite_matrix(Matrix  M)
{
  Matrix mv;
  register int i,j,k;

  mmode(MVIEWING);
  getmatrix(mv);

  /* form product mv * mp .... viewing followed by projection */
  for (i=0; i<4; i++) {
    for (k=0; k<4; k++) {
      M[i][k] = 0.0;
      for (j=0; j<4; j++)  M[i][k] += mv[i][j] * mp[j][k];
    }
```

```
  }
}

/*****************************************************************************/
/* Tipically, both functions above should be called together, and this is  */
/* what this function does...                                              */
/*****************************************************************************/
void prepareMatrix(Matrix M)
{
  setMatrices();
  get_composite_matrix(M);        /* View x Projection matrix */

}

/************** World to screen coordinates *********************************/
void world2screen(Matrix M, float x, float y, float z, float *sx, float *sy)
{
  float v[4];
  register int i;
  long xs = xsize, ys = ysize;

  for (i=0; i<4; i++) v[i] = x*M[0][i]+y*M[1][i]+z*M[2][i]+M[3][i];

  if (v[3]==0.0) {
    Bug("w2s: v[3] = 0.0");
    fprintf(stderr, "V (%f, %f, %f, %f)\n", v[0], v[1], v[2], v[3]);
    return;
    }
  *sx = ((float)xs/2.0)*(v[0]+1.0)-.5;
  *sy = ((float)ys/2.0)*(v[1]+1.0)-.5;
}

/************* Returns the distance between 2 points ************************/
float dist (float *pnt1, float *pnt2)
{
  return (fsqrt((pnt1[0]-pnt2[0]) * (pnt1[0]-pnt2[0]) +
```

```
            (pnt1[1]-pnt2[1]) * (pnt1[1]-pnt2[1])));
}

float hdist (float *pnt1, float *pnt2)
{
  return (fabs(pnt1[0]-pnt2[0]));
}

float vdist (float *pnt1, float *pnt2)
{
  return (fabs(pnt1[1]-pnt2[1]));
}

/* Matrix utilities */
/*****************************************************************************/
void copy_matrix (Matrix to, Matrix from)
{
  register int i, j;

  for (i=0; i<4; i++)
    for (j=0; j<4; j++)
      to[i][j] = from[i][j];
}

/*****************************************************************************/
void print_matrix(char *str, Matrix m)
{
  register int i, j;

  printf("Print matrix: %s\n", str);
  for (i=0; i<4; i++) {
    for (j=0; j<4; j++) {
      printf("%3.5f ",m[i][j]);
    }
    printf("\n");
  }
```

```
}

/*****************************************************************************/
void setPolarTwist(float val)
{
  left_parameter[POLAR_TWIST].value =
    right_parameter[POLAR_TWIST].value = val;
}
/*****************************************************************************/
void setPolarAzimuth(float val)
{
  left_parameter[POLAR_AZIM].value =
    right_parameter[POLAR_AZIM].value = val;
}
/*****************************************************************************/
void setPolarInc(float val)
{
  left_parameter[POLAR_INC].value =
    right_parameter[POLAR_INC].value = val;
}

/*****************************************************************************/
void addDist2Face(float delta)
{
  sc += delta;
}

/*****************************************************************************/
void addXoff(float delta)
{
  left_parameter[HEAD_Xoff].value =
    right_parameter[HEAD_Xoff].value += delta;
  left_parameter[HEAD_Xoff].changed =
    right_parameter[HEAD_Xoff].changed = 1;
}
```

```
/*****************************************************************************/
void addYoff(float delta)
{
  left_parameter[HEAD_Yoff].value =
    right_parameter[HEAD_Yoff].value += delta;
  left_parameter[HEAD_Yoff].changed =
    right_parameter[HEAD_Yoff].changed = 1;
}

/*****************************************************************************/
void addZoff(float delta)
{
  left_parameter[HEAD_Zoff].value =
    right_parameter[HEAD_Zoff].value += delta;
  left_parameter[HEAD_Zoff].changed =
    right_parameter[HEAD_Zoff].changed = 1;
}

/*****************************************************************************/
void addPolarTwist(float delta)
{
  left_parameter[POLAR_TWIST].value = right_parameter[POLAR_TWIST].value += delta;
  left_parameter[POLAR_TWIST].changed = right_parameter[POLAR_TWIST].changed = 1;
}

/*****************************************************************************/
void addPolarAzimuth(float delta)
{
  left_parameter[POLAR_AZIM].value = right_parameter[POLAR_AZIM].value += delta;
  left_parameter[POLAR_AZIM].changed = right_parameter[POLAR_AZIM].changed = 1;
}

/*****************************************************************************/
void addPolarInc(float delta)
{
  left_parameter[POLAR_INC].value = right_parameter[POLAR_INC].value += delta;
```

```
 left_parameter[POLAR_INC].changed = right_parameter[POLAR_INC].changed = 1;
}

/*****************************************************************************/
void saveTexCoord(char *filename, int np)
{
 FILE *f;
 char temp[MAX_NAME], param_file_name[MAX_NAME];
 int i;
 Matrix M;
 float p[2],width,height;

 calcTextureCoords(left_tex,right_tex);
 sprintf(temp,"%s.tex",filename);
 unlink(temp);
 if ((f = fopen(temp, "w")) == NULL) {
  fprintf(stderr, "Can't open texture file '%s' for writing!", temp);
  return;
 }
 fprintf(f, "Num of points: %d\n",np);
 fprintf(f,"Region: %f, %f, %f, %f\n", left_model, top_model,
         right_model, bottom_model);
 width = (floor)(right_model-left_model);
 height = (floor)(top_model-bottom_model);
 prepareMatrix(M);
 for(i=1; i<=np; i++) {
  world2screen(M, left_pt[i][0], left_pt[i][1], left_pt[i][2], &p[0], &p[1]);
  p[0] = (p[0]-left_model)/width;
  p[1] = (p[1]-bottom_model)/height;
  fprintf(f, "%d\t%f, %f\n",i,p[0], p[1]);
  world2screen(M, right_pt[i][0], right_pt[i][1], right_pt[i][2], &p[0], &p[1]);
  p[0] = (p[0]-left_model)/width;
  p[1] = (p[1]-bottom_model)/height;
  fprintf(f, "%d\t%f, %f\n",i,p[0],p[1]);
 }
 setPolarInc(90.0);
```

```
  setPolarTwist(0.0);
  setPolarAzimuth(0.0);
  sc = 0.35; /* Usually it is around this number... */
  prepareMatrix(M);
  fprintf(f, "# Normalized projection of selected points of model:\n");
  for(i=49; i<=70; i++) {
    world2screen(M, left_pt[i][0], left_pt[i][1], left_pt[i][2], &p[0], &p[1]);
    fprintf(f, "%d\t%d, %d\n",i,(int)p[0], (int)p[1]);
    world2screen(M, right_pt[i][0], right_pt[i][1], right_pt[i][2], &p[0], &p[1]);
    fprintf(f, "%d\t%d, %d\n",i,(int)p[0], (int)p[1]);
  }
  i = 287;
  world2screen(M, left_pt[i][0], left_pt[i][1], left_pt[i][2], &p[0], &p[1]);
  fprintf(f, "%d\t%.0f, %0f\n",i,p[0]+.5, p[1]+.5);
  world2screen(M, right_pt[i][0], right_pt[i][1], right_pt[i][2], &p[0], &p[1]);
  fprintf(f, "%d\t%0f, %0f\n",i,p[0]+.5, p[1]+.5);

  fclose(f);
    sprintf(param_file_name, "%s.%d.prm",
out_base_name,(int)MOVIE_getCurrentFrameNo(movie));
  restoreParameters(param_file_name,left_parameter,right_parameter,TRUE);
}

/*****************************************************************************/
void restoreTexCoord(char *filename,
            float left_pt[][2], float right_pt[][2],
            float left_tex[][2], float right_tex[][2])
{
  FILE *f;
  char temp[MAX_NAME],line[MAX_LINE];
  int i, j, np;
  float width,height;

  if ((f = fopen(filename, "r")) == NULL) {
    sprintf(temp,"%s.tex",filename);
    if ((f = fopen(temp, "r")) == NULL) {
```

```
    fprintf(stderr, "Can't open texture file '%s' for reading!", filename);
    return;
   }
  }

  fgets(line,MAX_LINE,f);
  sscanf(line, "Num of points: %d\n",&np);
  fgets(line,MAX_LINE,f);
  sscanf(line, "Region: %f, %f, %f, %f\n",
        &left_model, &top_model, &right_model, &bottom_model);

  width = (floor)(right_model-left_model);
  height = (floor)(top_model-bottom_model);
  for(i=1; i<=np; i++) {
   fgets(line,MAX_LINE,f);
   sscanf(line, "%d\t%f, %f",&j,&left_tex[i][0],&left_tex[i][1]);
   fgets(line,MAX_LINE,f);
   sscanf(line, "%d\t%f, %f",&j,&right_tex[i][0],&right_tex[i][1]);
  }
  fgets(line,MAX_LINE,f); /* Remark line */
  while (!feof(f)) {
   fgets(line,MAX_LINE,f);
   sscanf(line, "%d",&j);
   sscanf(line, "%*d\t%f, %f",&left_pt[j][0],&left_pt[j][1]);
   fgets(line,MAX_LINE,f);
   sscanf(line, "%*d\t%f, %f",&right_pt[j][0],&right_pt[j][1]);
  }

  fclose(f);
}

/*************************************************************************/
/* Filename should be added ".mth1" and ".mth2"                          */
/* Then, inside of mouth (without lips) coordinates are saved in .mth1   */
/* file, and outside of the moith is saved in .mth2 file                 */
/*************************************************************************/
```

```
void saveInMth(char *filename)
{
 FILE *f;
 char temp[MAX_NAME];

 /* Save inside of mouth */
 sprintf(temp, "%s.mth1", filename);
 unlink(temp);
 if ((f = fopen(temp, "w")) == NULL) {
  fprintf(stderr, "Can't open mth param file '%s' for writing!", temp);
  return;
 }
 fprintf(f, "%f, %f\n",right_tex[287][0], right_tex[287][1]); /* 0 */
 fprintf(f, "%f, %f\n",right_tex[63][0],  right_tex[63][1]);  /* 1 */
 fprintf(f, "%f, %f\n",right_tex[62][0],  right_tex[62][1]);  /* 2 */
 fprintf(f, "%f, %f\n",right_tex[61][0],  right_tex[61][1]);  /* 3 */
 fprintf(f, "%f, %f\n",left_tex[62][0],   left_tex[62][1]);   /* 4 */
 fprintf(f, "%f, %f\n",left_tex[63][0],   left_tex[63][1]);   /* 5 */
 fprintf(f, "%f, %f\n",left_tex[287][0],  left_tex[287][1]);  /* 6 */
 fprintf(f, "%f, %f\n",left_tex[60][0],   left_tex[60][1]);   /* 7 */
 fprintf(f, "%f, %f\n",left_tex[59][0],   left_tex[59][1]);   /* 8 */
 fprintf(f, "%f, %f\n",left_tex[58][0],   left_tex[58][1]);   /* 9 */
 fprintf(f, "%f, %f\n",right_tex[59][0],  right_tex[59][1]);  /* 10 */
 fprintf(f, "%f, %f\n",right_tex[60][0],  right_tex[60][1]);  /* 11 */
 fclose(f);

 /* Save outside of mouth */
 sprintf(temp, "%s.mth2", filename);
 unlink(temp);
 if ((f = fopen(temp, "w")) == NULL) {
  fprintf(stderr, "Can't open mth param file '%s' for writing!", temp);
  return;
 }
 fprintf(f, "%f, %f\n",right_tex[70][0], right_tex[70][1]);   /* 0 */
 fprintf(f, "%f, %f\n",right_tex[69][0],  right_tex[69][1]);  /* 1 */
 fprintf(f, "%f, %f\n",right_tex[68][0],  right_tex[68][1]);  /* 2 */
```

```
fprintf(f, "%f, %f\n",right_tex[67][0],  right_tex[67][1]);  /* 3 */
fprintf(f, "%f, %f\n",left_tex[68][0],   left_tex[68][1]);   /* 4 */
fprintf(f, "%f, %f\n",left_tex[69][0],   left_tex[69][1]);   /* 5 */
fprintf(f, "%f, %f\n",left_tex[70][0],  left_tex[70][1]);    /* 6 */
fprintf(f, "%f, %f\n",left_tex[51][0],   left_tex[51][1]);   /* 7 */
fprintf(f, "%f, %f\n",left_tex[50][0],   left_tex[50][1]);   /* 8 */
fprintf(f, "%f, %f\n",left_tex[49][0],   left_tex[49][1]);   /* 9 */
fprintf(f, "%f, %f\n",right_tex[50][0],  right_tex[50][1]);  /* 10 */
fprintf(f, "%f, %f\n",right_tex[51][0],  right_tex[51][1]);  /* 11 */
fclose(f);

/* Save external mouth */
sprintf(temp, "%s.mth3", filename);
unlink(temp);
if ((f = fopen(temp, "w")) == NULL) {
  fprintf(stderr, "Can't open mth param file '%s' for writing!", temp);
  return;
}
fprintf(f, "%f, %f\n",right_tex[44][0], right_tex[44][1]);   /* 0 */
fprintf(f, "%f, %f\n",right_tex[73][0],  right_tex[73][1]);  /* 1 */
fprintf(f, "%f, %f\n",right_tex[72][0],  right_tex[72][1]);  /* 2 */
fprintf(f, "%f, %f\n",right_tex[71][0],  right_tex[71][1]);  /* 3 */
fprintf(f, "%f, %f\n",left_tex[72][0],   left_tex[72][1]);   /* 4 */
fprintf(f, "%f, %f\n",left_tex[73][0],   left_tex[73][1]);   /* 5 */
fprintf(f, "%f, %f\n",left_tex[44][0],  left_tex[44][1]);    /* 6 */
fprintf(f, "%f, %f\n",left_tex[43][0],   left_tex[43][1]);   /* 7 */
fprintf(f, "%f, %f\n",left_tex[42][0],   left_tex[42][1]);   /* 8 */
fprintf(f, "%f, %f\n",left_tex[41][0],   left_tex[41][1]);   /* 9 */
fprintf(f, "%f, %f\n",right_tex[42][0],  right_tex[42][1]);  /* 10 */
fprintf(f, "%f, %f\n",right_tex[43][0],  right_tex[43][1]);  /* 11 */
fclose(f);

}
/*************************************************************************/
int int_comp (const void *i1, const void *i2)
```

```
{
 return (*((int *)i1) - *((int *)i2));
}

/*****************************************************************************/
int readKeyFrameIndices (char *filename, int *key_frames, int *init_frames)
{
 FILE *f;
 int i = 1;
 char temp[MAX_LINE];

 key_frames[0] = init_frames[0] = 0;
 sprintf(temp,"%s.kf",filename);
 if ((f = fopen(temp, "r")) == NULL) {
  return (FALSE);
 }

 while (fgets(temp,MAX_LINE,f) && !feof(f)) {
  sscanf(temp, "%d", &key_frames[i]);
  sscanf(temp, "%d", &init_frames[i++]);
 }
 fclose(f);
 key_frames[0] = i-1;
 qsort (key_frames+1,key_frames[0],sizeof(int),int_comp);

 sprintf(temp,"%s.if",filename);
 if ((f = fopen(temp, "r")) == NULL) {
  return (FALSE);
 }
 while (fgets(temp,MAX_LINE,f) && !feof(f)) {
  sscanf(temp, "%d", &init_frames[i++]);
 }
 fclose(f);

 init_frames[0] = i-1;
```

```
  qsort (init_frames+1,init_frames[0],sizeof(int),int_comp);
  printf("%d init frames: ",init_frames[0]);
  for (i=1; i <= init_frames[0]; i++)
    printf ("%d ", init_frames[i]);
  printf("\n%d key frames: ",key_frames[0]);
  for (i=1; i <= key_frames[0]; i++)
    printf ("%d ", key_frames[i]);
  printf("\n");
  return (TRUE);
}

/*****************************************************************************/
void restoreInMth(char *filename, float into[12][2])
{
  FILE *f;
  int i;
  char temp[MAX_LINE];

  if ((f = fopen(filename, "r")) == NULL) {
    return;
  }

  for (i=0; i < 12; i++) {
    fgets(temp,MAX_LINE,f);
    sscanf(temp, "%f, %f", &into[i][0], &into[i][1]);
  }
  fclose(f);
}

/*****************************************************************************/
void swapParameters(void)
{
  int j;

  if (which_image == SIDE_IMG) {
```

```
    for (j=0; j < NUM_SAVED; j++) {
      saved_front_params[0][j] = VAL(left_parameter[save_those[j]]);
      saved_front_params[1][j] = VAL(right_parameter[save_those[j]]);
      VAL(left_parameter[save_those[j]]) = saved_side_params[0][j];
      VAL(right_parameter[save_those[j]]) = saved_side_params[1][j];
    }
  }
  else {
    for (j=0; j < NUM_SAVED; j++) {
      saved_side_params[0][j] = VAL(left_parameter[save_those[j]]);
      saved_side_params[1][j] = VAL(right_parameter[save_those[j]]);
      VAL(left_parameter[save_those[j]]) = saved_front_params[0][j];
      VAL(right_parameter[save_those[j]]) = saved_front_params[1][j];
    }
  }
}

/*************************************************************************/
/* Converts Gimg (unsigned long (RGB)) image to char *result, which is */
/* The weighted average of the input image. Expects result memory to  */
/* be allocated.                                                      */
/*************************************************************************/
void longImg2charVec (u_long *img, int sx, int sy, int width,int  height, int xsize, char *result)
{
  register int y, x, dx = xsize - width;
  img += sy*xsize + sx;

  for (y=0; y < height; y++) {
    for (x=0; x < width; x++) {
      *result = (char)greyColor(*img);
      result++;
      img++;
    }
  img += dx;
  }
```

```
}

/*****************************************************************************/
void findRegion(int from_vertex,
        int to_vertex,
        float *left,
        float *top,
        float *right,
        float *bottom,
        float w_scale_factor,
        float h_scale_factor)
{
 float *u, *d, *l, *r;
 Matrix M;
 float temp;
 float max_x, max_y, min_x, min_y;
 int size, cor,j, max_x_i, max_y_i,min_x_i,min_y_i;
 float width, height,lp[2],rp[2];

 max_x = -99999.0, max_y = -99999.9;
 min_x = 99999.0,  min_y = 99999.9;
 prepareMatrix(M);
 for (j=from_vertex; j <= to_vertex; j++) {
  world2screen(M, left_pt[j][0], left_pt[j][1], left_pt[j][2], &lp[0], &lp[1]);
  world2screen(M, right_pt[j][0], right_pt[j][1], right_pt[j][2], &rp[0], &rp[1]);
  if (GT(lp[0], max_x))  { max_x = lp[0]; l = left_pt[j];max_x_i=j;}
  if (GT(rp[0], max_x)) { max_x = rp[0]; l = right_pt[j];max_x_i=j;}
  if (LT(lp[0], min_x))  { min_x = lp[0]; r = left_pt[j];min_x_i=j;}
  if (LT(rp[0], min_x)) { min_x = rp[0]; r = right_pt[j];min_x_i=j;}

  if (GT(lp[1], max_y))  { max_y = lp[1]; u = left_pt[j];max_y_i=j;}
  if (GT(rp[1], max_y)) { max_y = rp[1]; u = right_pt[j];max_y_i=j;}
  if (LT(lp[1], min_y))  { min_y = lp[1]; d = left_pt[j];min_y_i=j;}
  if (LT(rp[1], min_y)) { min_y = rp[1]; d = right_pt[j];min_y_i=j;}
 }
```

```
  world2screen(M, r[0], r[1], r[2], left, &temp);
  world2screen(M, u[0], u[1], u[2], &temp, top);
  world2screen(M, d[0], d[1], d[2], &temp, bottom);
  world2screen(M, l[0], l[1], l[2], right, &temp);

  width    = (*right) - (*left) + 1;
  height   = (*top) - (*bottom) + 1;
  *left   -= width * w_scale_factor;
  *right  += width * w_scale_factor;
  *top    += height * h_scale_factor;
  *bottom -= height * h_scale_factor;

  *left   = MAX(1,(*left));
  *right  = MIN(MOVIE_frameWidth(movie)-1,(*right));
  *top    = MIN(MOVIE_frameHeight(movie)-1,(*top));
  *bottom = MAX(1,(*bottom));
}

/*****************************************************************************/
void setModelRegion(void)
{
  findRegion(1,np,
             &left_model,  &top_model,
             &right_model, &bottom_model,
             0.0,0.0);

  left_model   = floor(left_model);
  right_model  = floor(right_model+.5);
  bottom_model = floor(bottom_model);
  top_model    = floor(top_model+.5);
}

Parameter temp_left[MAX_PARAMETERS],  temp_right[MAX_PARAMETERS];

/*****************************************************************************/
```

```
int restoreLipsParameters(int frame_no)
{
 int i, prm;
 float l1,l2,l3,r1,r2,r3;
 char temp[MAX_NAME];
 int which[] = {JAW_ROTATION,MTH_Yscl,RSE_UPLIP,LWRLIP_FTUCK,MTH_INTERP};

 sprintf(temp,"%s.%d.prm",out_base_name, frame_no);
 if (!restoreParameters(temp,temp_left, temp_right,TRUE)) return (FALSE);
 for (i=0; i < XtNumber(which); i++) {
  prm = which[i];
  VAL(left_parameter[prm]) = VAL(temp_left[prm]);
  VAL(right_parameter[prm]) = VAL(temp_right[prm]);
 }
 recompute_face();
 return (TRUE);
}

/*****************************************************************************/
void restoreModel(char *filename)
{
 char temp[MAX_LINE];
 FILE *f;
 int i;
 float *p;
 if ((f = fopen(filename, "r")) == NULL) {
  printf("%s not found!\n",filename);
  return;
 }

 fgets (temp, MAX_LINE, f);
 for (i=1; i <= np; i++) {
  fgets (temp, MAX_LINE, f);
  p = left_inpts[i];
  sscanf(temp, "%*d\t%f\t%f\t%f", &p[0], &p[1], &p[2]);
```

```
    fgets (temp, MAX_LINE, f);
    p = right_inpts[i];
    sscanf(temp, "%*d\t%f\t%f\t%f", &p[0], &p[1], &p[2]);
  }
  fclose(f);
}

/*****************************************************************************/
void restoreAllParameters(char *filename,
          int n_frames,
          Parameter **left_array,
          Parameter **right_array)
{
  int i, j,prm;
  char temp[MAX_NAME];

  for (i=0; i<n_frames; i++) {
    MALLOC(left_array[i],MAX_PARAMETERS,Parameter);
    MALLOC(right_array[i],MAX_PARAMETERS,Parameter);
    bzero(left_array[i],MAX_PARAMETERS*sizeof(Parameter));
    bzero(right_array[i],MAX_PARAMETERS*sizeof(Parameter));
    sprintf(temp, "%s.%d.prm", filename, i);
    restoreParameters(temp,left_array[i],right_array[i],FALSE);

    if (i==0 && (n_frames > 1)) {
      for (j=0; j < NUM_ACTIVE_MTH_PARAMS;  j++) {
          prm = active_mth_params[j];
          MAX_VAL(left_array[0][prm])   = VAL(left_array[0][prm]);
          MIN_VAL(left_array[0][prm])   = VAL(left_array[0][prm]);
          MAX_VAL(right_array[0][prm])  = VAL(right_array[0][prm]);
          MIN_VAL(right_array[0][prm])  = VAL(right_array[0][prm]);
      }
      }
    else for (j=0; j< NUM_ACTIVE_MTH_PARAMS;  j++) {
      prm = active_mth_params[j];
```

```
    if (GT(VAL(left_array[i][prm]),MAX_VAL(left_array[0][prm])))
        MAX_VAL(left_array[0][prm])   = VAL(left_array[i][prm]);
    if (LT(VAL(left_array[i][prm]), MIN_VAL(left_array[0][prm])))
        MIN_VAL(left_array[0][prm])   = VAL(left_array[i][prm]);
    if (GT(VAL(right_array[i][prm]), MAX_VAL(right_array[0][prm])))
        MAX_VAL(right_array[0][prm])  = VAL(right_array[i][prm]);
    if (LT(VAL(right_array[i][prm]), MIN_VAL(right_array[0][prm])))
        MIN_VAL(right_array[0][prm])  = VAL(right_array[i][prm]);
   }
  }
}

/*****************************************************************************/
void calcProjection(float left_pts[][2], float right_pts[][2])
{
 register int j;
 float x, y, osc= sc, oinc = VAL(left_parameter[POLAR_INC]),
      otwist = VAL(left_parameter[POLAR_TWIST]),
      oazim = VAL(left_parameter[POLAR_AZIM]);
 Matrix M;

 setPolarInc(90.0);
 setPolarTwist(0.0);
 setPolarAzimuth(0.0);
 sc = 0.35; /* Usually it is around this number... */
 recompute_face();
 prepareMatrix(M);
 for (j=1; j <= np; j++) {
  world2screen(M, left_pt[j][0], left_pt[j][1], left_pt[j][2], &x, &y);
  left_pts[j][0]  = /*floor*/(x);
  left_pts[j][1]  = /*floor*/(y);
  world2screen(M, right_pt[j][0], right_pt[j][1], right_pt[j][2], &x, &y);
  right_pts[j][0] = /*floor*/(x);
  right_pts[j][1] = /*floor*/(y);
 }
```

```
  sc = osc;
  setPolarInc(oinc);
  setPolarTwist(otwist);
  setPolarAzimuth(oazim);

}

/*****************************************************************************/
void calcAvg (float left_pts[][2],
              float right_pts[][2],
              float *mid_x,
              float *mid_y,
              index_st *pts_index)
{
  float x=0., y=0.;
  int i;

  for (i=0; i< NUM_MTH_PTS; i++) {
    if (pts_index[i].side == LEFT_PARAMS) {
      x += left_pts[pts_index[i].pnt][0];
      y += left_pts[pts_index[i].pnt][1];
    }
    else {
      x += right_pts[pts_index[i].pnt][0];
      y += right_pts[pts_index[i].pnt][1];
    }
  }

  *mid_x = x / NUM_MTH_PTS;
  *mid_y = y / NUM_MTH_PTS;
}

/*****************************************************************************/
void setMinMax (int *array)
{
```

```
int nf = array[0], i, j, k, pnt, first = 1,prm;
char temp[MAX_NAME];
index_st *indices;
float p[2], range;

for (i=1; i <= nf; i++) {
 sprintf(temp,"%s.%d.prm",out_base_name, array[i]);

 /*   printf("Checking %s\n",temp);*/
 if (!restoreParameters(temp,temp_left, temp_right,TRUE)) {
  printf("Warning: file %s doesn't exist!!!\n", temp);
  if (i==first) first++;
 }
 else {
  if (i==first) {/* First file sets all max and min values! */
      for (j=0; j< NUM_ACTIVE_MTH_PARAMS; j++) {
       prm = active_mth_params[j];
       MAX_VAL(left_parameter[prm])  = VAL(temp_left[prm]);
       MIN_VAL(left_parameter[prm])  = VAL(temp_left[prm]);
       MAX_VAL(right_parameter[prm]) = VAL(temp_right[prm]);
       MIN_VAL(right_parameter[prm]) = VAL(temp_right[prm]);
      }
  }
  else {
      for (j=0; j< NUM_ACTIVE_MTH_PARAMS; j++) {
       prm = active_mth_params[j];
       if (GT(VAL(temp_left[prm]),MAX_VAL(left_parameter[prm])))
         MAX_VAL(left_parameter[prm])  = VAL(temp_left[prm]);
       if (LT(VAL(temp_left[prm]), MIN_VAL(left_parameter[prm])))
         MIN_VAL(left_parameter[prm])  = VAL(temp_left[prm]);
       if (GT(VAL(temp_right[prm]), MAX_VAL(right_parameter[prm])))
         MAX_VAL(right_parameter[prm])  = VAL(temp_right[prm]);
       if (LT(VAL(temp_right[prm]), MIN_VAL(right_parameter[prm])))
         MIN_VAL(right_parameter[prm])  = VAL(temp_right[prm]);
      }
```

```
        }
      }
    }
      printf("Total of %d keyframes\n", nf);
      printf("Min and Max values:\n----------\n");

    for (j=0; j< NUM_ACTIVE_MTH_PARAMS; j++) {
     prm = active_mth_params[j];
     if (MAX_VAL(right_parameter[prm]) == MIN_VAL(right_parameter[prm])) {
      sprintf(temp,"%s.0.prm",out_base_name);
      restoreParameters(temp,temp_left, temp_right,FALSE);
      MIN_VAL(right_parameter[prm]) = MIN_VAL(temp_right[prm]);
      MIN_VAL(left_parameter[prm]) = MIN_VAL(temp_left[prm]);
      MAX_VAL(right_parameter[prm]) = MAX_VAL(temp_right[prm]);
      MAX_VAL(left_parameter[prm]) = MAX_VAL(temp_left[prm]);
     }
     range = (MAX_VAL(left_parameter[prm])-MIN_VAL(left_parameter[prm]))*range_limit;
     MIN_VAL(left_parameter[prm]) -= range;
     MAX_VAL(left_parameter[prm]) += range;
     range=(MAX_VAL(right_parameter[prm])-MIN_VAL(right_parameter[prm]))*range_limit;
     MIN_VAL(right_parameter[prm]) -= range;
     MAX_VAL(right_parameter[prm]) += range;
     printf("(%d) %s: left min = %f, left max = %f\n\t\t right min = %f, right max = %f\n",
       prm,
       left_parameter[prm].box_label,
       MIN_VAL(left_parameter[prm]),
       MAX_VAL(left_parameter[prm]),
       MIN_VAL(right_parameter[prm]),
       MAX_VAL(right_parameter[prm]));
    }
}

/*****************************************************************************/
int intInArray(int val, int *array)
{
```

```
        int i, n = array[0];

  for (i=1; i<=n; i++)
    if (val == array[i]) return (TRUE);

  return (FALSE);
}

/*****************************************************************************/
float calcBeard (float p288[2])
{
  float p16[2], distance,sb;
  Matrix M;

  prepareMatrix(M);
  world2screen(M,left_pt[16][0],left_pt[16][1],left_pt[16][2],&p16[0],  &p16[1]);
  world2screen(M,left_pt[288][0],left_pt[288][1],left_pt[288][2],&p288[0],  &p288[1]);
  distance = dist (p16,p288);
}
/*****************************************************************************
 * Module: gimg.h - interface for gimg.c - image utils
 * By: Ranco
 * History:
 *  13.4.94: creation.
 *  2nd version (MOVIE based): 24.4.95
*****************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <strings.h>
#include <fcntl.h>
#include <getopt.h>
#include <gl/gl.h>
#include <gl/device.h>
#include <gl/image.h>
```

```
/* defines for image file manipulation */

#define GCMAP 1
#define GRGB3
#define GRGBA4

typedef unsigned long   MPIXEL;
typedef unsigned char   PELEMENT;

#define REDM 0x000000ff
#define GRNM 0x0000ff00
#define BLUM 0x00ff0000
#define ALPM 0xff000000
#define red(x) ((x)&REDM)
#define green(x) (((x)&GRNM)>>8)
#define blue(x) (((x)&BLUM)>>16)
#define alpha(x) (((x)&ALPM)>>24)

typedef struct {
   unsigned short xsiz, ysiz, zsiz;
   char *buf;
   unsigned short type;
} GimgRec, *Gimg;

#define ImgXsize(i)     ((i)->xsiz)          /* num colomns   */
#define ImgYsize(i)     ((i)->ysiz)          /* num rows      */
#define ImgZsize(i)     ((i)->zsiz)          /* # color bands */
#define ImgRast(i)      ((i)->buf)
#define ImgType(i)      ((i)->type)
#define ImgPixRGBadd(i, x, y) ((i)->buf + ((y)*ImgXsize(i)+(x)) *sizeof(MPIXEL))
#define ImgPixCMAPadd(i, x, y) ((i)->buf+ ((y)*ImgXsize(i)+(x)) *sizeof(short))
#define ImgPixRGB(i, x, y)  ( ((MPIXEL *)(i)->buf) [(y)*ImgXsize(i)+(x)])
#define ImgPixCMAP(i, x, y) ( ((short *)(i)->buf) [(y)*ImgXsize(i)+(x)])
#define ImgInRange(i, x, y) ((x>=0.0)&&(y>=0.0)&& \
```

```
(x<(double)ImgXsize(i))&&(y<(double)ImgYsize(i)))
Gimg GimgCreate(short xsize, short ysize, int zsize);
int GimgInit(Gimg img, short xsize, short ysize, int zsize);
int GimgFree(Gimg img);
int GimgDel(Gimg img);
Gimg GimgRead(char *f);
int GimgWrite(Ptr f, Gimg img);
void GimgDisp(Gimg img, short x0, short y0);
unsigned long GimgCalcSum(Gimg img);
void GimgClear(Gimg img);
Gimg GimgCreateSubCopy(Gimg orig, int x0, int y0, int width, int height);
Gimg GimgResize (Gimg image, int w, int h, int sample);
void GimgAvg(Gimg img, long *r, long *g, long *b, int include_zeros);
void GimgMask(Gimg img, long r, long g, long b, int x0, int y0, int width, int height);
void GimgSetAlpha (Gimg img, long a);
int GimgFlip(Gimg img);
int GimgInterlace(Gimg origin, Gimg Destination);
int GimgDeInterlace(Gimg origin, Gimg Destination);
/*****************************************************************************
 * Module: image.c
 * By: Ranco
 * History:
 *  18.8.94: creation.
*****************************************************************************/
#include "mv.h"
#include "defs.h"

extern long bgc;

/*****************************************************************************/
void putimage (void)
{
 MOVIE_showFrame(movie,(MVframe)0);
}
```

```
/*****************************************************************************
 * Module: io.c
 * By: Ranco
 * History:
 *  18.8.94: creation.
 *  2nd version (MOVIE based): 24.4.95
*****************************************************************************/
#include "list.h"

/*****************************************************************************/
/* Load points info from filename, into PntList Pl                           */
/*****************************************************************************/
void loadPts (char *filename, PntList *Pl)
{
 FILE *fp;
 int i, dummy, count = 0;
 int x, y, res;
 char name[MAX_NAME];
 Point P;

 if ((fp = fopen (filename, "r")) == NULL) {
  printf("%s\n",filename);
  PRINT_ERR("Can't find data file\n");
 }

 while(!feof(fp)) {
  if ((res = fscanf(fp, "%d %d %d", &dummy, &x, &y)) != 3 ||
      (dummy != count+1)) {
   if (res == -1) break;
   PRINT_ERR("Wrong points file format!\n");
  }
  count++;
 }

 *Pl = createPntList();
 fseek(fp,0,0);
```

```
 for (i = 0; i < count; i++) {
  if (fscanf(fp, "%d %d %d", &dummy, &x, &y) != 3 ||
     (dummy != i+1)) {
   PRINT_ERR("Wrong points file format!\n");
  }
  P.x = (double)x;
  P.y = (double)y;
  addPoint(*Pl, &P);
 }
 fclose(fp);
}

/*****************************************************************************
 * Module: list.h - interface for list.c
 * By: Ranco
 * History:
 *  16.2.94: creation.
*****************************************************************************/
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "defs.h"

#define NUM_OF_PNTS(pl)    ((pl)->Size)
#define POINT(pl,p)        ((pl)->Points[p])
#define NUM_OF_LINES(ll)   ((ll)->Size)
#define LINE(ll,p)         ((ll)->Lines[p])
#define LENG(l)            ((l).len)
#define SOURCE(l)          ((l).P1)
#define DEST(l)            ((l).P2)
#define XCOR(p)            ((p).x)
#define YCOR(p)            ((p).y)

/*  Some point operations */
#define SIZE(p)            (XCOR(p)*XCOR(p)+YCOR(p)*YCOR(p))
#define SetPoint(p,a,b)    {(p).x = (a); (p).y = (b); }
```

```
#define GetPoint(p,r)       {(r).x=(p).x; (r).y=(p).y; }
#define AddPoints(p,p1,p2)  {(p).x = (p1).x+(p2).x; (p).y = (p1).y+(p2).y; }
#define SubPoints(p,p1,p2)  {(p).x = (p1).x-(p2).x; (p).y = (p1).y-(p2).y; }
#define MulPoint(p,c)       {(p).x *= (float) (c); (p).y *= (float) (c); }
#define DivPoint(p,c)       {(p).x /= (float) (c); (p).y /= (float) (c); }
#define DotProduct(p1,p2)   ((p1).x*(p2).x+(p1).y*(p2).y)

#define SetLine(l,p1,p2)    {(l).P1.x = (p1)->x; (l).P1.y = (p1)->y; \
                            (l).P2.x = (p2)->x; (l).P2.y = (p2)->y; }
typedef struct {
  float x, y;
} Point;
typedef struct {
   int Size;
   Point *Points;
} PntList_st;
typedef PntList_st *PntList;
typedef struct {
  Point P1, P2;
  int len;
} Line;
typedef struct {
   int Size;
   Line *Lines;
   } LineList_st;
typedef LineList_st *LineList;

/* Prototypes */
PntList createPntList (void) ;
void freePntList (PntList l) ;
PntList addPoint (PntList l, Point *p) ;
LineList createLineList (void) ;
void freeLineList (LineList l) ;
LineList addLine (LineList l, Point *P1, Point *P2);

/* ---------------------------------------------------------------------- */
```

```
/* File: mv.h                                                    */
/*                                                               */
/* Include file for the interface to movie (mv) library          */
/*                                                               */
/* Written by: Ranco                                             */
/* Creation: Wed, 5/4/95                                         */
/* ------------------------------------------------------------- */
#ifndef __MV_H
#define __MV_H

#include <dmedia/cl.h>
#include <dmedia/cl_cosmo.h>
#include <movie.h>
#include "gimg.h"
#include "defs.h"
#include <audiofile.h>

#define MOVIE_ERR       0
#define MOVIE_OK        1


typedef struct {
 char file_name[MAX_NAME];
 MVid movie, imagetrack, audiotrack;
 int width, height; /* Size of a movie frame */
 int interlace;
 MVframe first, last;  /* First and last frame's indices, for playback */
 MVframe num_frames;
 MVframe current_frame_no;
 double frame_rate;
 double audio_rate;
 int audio_width;
 char compression[20];
 CLhandle compressor, decompressor;
 int cosmo_available;
 int paramBuf[10];
```

```
  int n_params;
  Gimg current_frame;
  int del_img;  /* a flag that indicate if current frame should be freed at the end */
  int orientation;
  int quality; /* Jpeg Quality (0 - 100) */
} Movie_st;

typedef Movie_st *Movie;

/********* MACROS ***********/
#define MOVIE_movieName(mv)          (mv->file_name)
#define MOVIE_movieTrack(mv)         (mv->movie)
#define MOVIE_imageTrack(mv)         (mv->imagetrack)
#define MOVIE_audiotrack(mv)         (mv->audiotrack)
#define MOVIE_frameWidth(mv)         (mv->width)
#define MOVIE_frameHeight(mv)        (mv->height)
#define MOVIE_numOfFrames(mv)        (mvGetTrackLength(mv->imagetrack))
#define MOVIE_frameRate(mv)          (mv->frame_rate)
#define MOVIE_audioRate(mv)          (mv->audio_rate)
#define MOVIE_audioWidth(mv)         (mv->audio_width)
#define MOVIE_firstFrame(mv)         (mv->first)
#define MOVIE_lastFrame(mv)          (mv->last)
#define MOVIE_getCurrentFrameNo(mv)  (mv->current_frame_no)
#define MOVIE_currentFrame(mv)       (mv->current_frame)
#define MOVIE_loadCurrentFrame(mv)   (MOVIE_getCurrentFrame(mv,mv->current_frame))
#define MOVIE_frameOrientation(mv)   (mv->orientation)
#define MOVIE_frameCompression(mv)   (mv->compression)
#define MOVIE_frameInterlace(mv)     (mv->interlace)
#define MOVIE_jpegQuality(mv)        (mv->quality)

/*****************************************************************************/

extern Movie movie;  /* Currently allow one movie in the application! */

/*** Prototypes ***/
Movie MOVIE_open (char *filename);
```

```
DMparams *MOVIE_init (Movie mv,
                      char *filename,
                      int width,
                      int height,
                      double frame_rate,
                      double audio_rate,
                      int orientation,
                      char *compression,
                      int interlace);
Movie MOVIE_create (char *filename,
                    int width,
                    int height,
                    double frame_rate,
                    double audio_rate,
                    int orientation,
                    char *compression,
                    int interlace,
                    int jpeg_quality);
int MOVIE_addFrame (Movie mv, MVframe index, Gimg frame, int size);
int MOVIE_copyFrames (Movie from_mv, Movie to_mv, MVframe from_index, MVframe
to_index, MVframe count);
int MOVIE_deleteFrames (Movie mv, MVframe index, MVframe count);
int MOVIE_getFrame (Movie mv, MVframe index, Gimg frame);
int MOVIE_setCurrentFrameNo(Movie  mv, MVframe index);
int MOVIE_getCurrentFrame (Movie mv, Gimg frame);
int MOVIE_addCurrentFrame (Movie mv, Gimg frame, int size);
int MOVIE_nextFrame (Movie mv);
int MOVIE_prevFrame (Movie mv);
int MOVIE_showFrame (Movie mv, MVframe frame_no);
int MOVIE_bind ();
int MOVIE_play (Movie mv);
int MOVIE_stop (Movie mv);
int MOVIE_home (Movie mv);
int MOVIE_end(Movie  mv);
int MOVIE_setStartFrame(Movie  mv, MVframe frame_no);
int MOVIE_setEndFrame(Movie  mv, MVframe frame_no);
```

```
int MOVIE_close (Movie mv);
size_t MOVIE_getFrameSize (Movie mv, int frame_no);
Gimg MOVIE_resize (Gimg image, int w, int h, int sample);
int MOVIE_addAudioFrame (AFfilehandle audioFile, Movie to_mv);
AFfilehandle MOVIE_createAudioTrack (char *audio_file_name, Movie to_mv);

#endif
/*****************************************************************************
 * Module: texture.c
 * By: Ranco
 * History:
 *  18.8.94: creation.
*****************************************************************************/
#include "texture.h"
#include "x_inc.h"
#include "general.h"
#include "main.h"
#include "vars.h"
#include "mv.h"

float texprops[] = {
  TX_MINFILTER, TX_POINT,
  TX_MAGFILTER, TX_POINT,
/*  TX_MINFILTER, TX_BILINEAR,
  TX_MAGFILTER, TX_BILINEAR,*/
  TX_WRAP, TX_CLAMP, TX_NULL};
float tevprops[] = {TV_DECAL, TV_NULL};

extern int texture_map;

/*****************************************************************************/
void textureMapping (int op)
{
  long xs,ys;
  Gimg sub_img;
  int width,height;
```

```
if (op == TEX_ON) {
  texture_map = TRUE;
  wire = FALSE;
  if (!show_coord) {
    MOVIE_loadCurrentFrame (movie);
  }

  calcTextureCoords(left_tex,right_tex);
  width = (int)(right_model-left_model);
  height = (int)(top_model-bottom_model);
                    s u b _ i m g =
GimgCreateSubCopy(MOVIE_currentFrame(movie),(int)left_model,(int)bottom_model,width,
height);

  zclear();
  if (which_image == SIDE_IMG) {
    xs = ImgXsize(side_img);
    ys = ImgYsize(side_img);
    texdef2d(1, 4, xs, ys, (u_long *)ImgRast(side_img), 0, texprops);
  }
  else {
    GimgSetAlpha(sub_img,255l);
    texdef2d(1, 4, width,height, (u_long *)ImgRast(sub_img), 0, texprops);
  }

  tevdef(1, 0, tevprops);
  tevbind(TV_ENV0, 1);
  texbind(TX_TEXTURE_0, 1);
  GimgDel(sub_img);
}
else {
  texture_map = FALSE;
  wire = TRUE;
  tevbind(TV_ENV0, 0);
  texbind(TX_TEXTURE_0,0);
  Refresh();
```

```
  }
  RedrawNeeded = TRUE;
}

/*************************************************************************/
void calcTextureCoords(float left_tex[][2], float right_tex[][2])
{
  register int j;
  float xs, ys,l=0.0,b=0.0;
  Matrix M;

  setModelRegion(); /* Set global left/right/top/bottom_model to the
                       current model's region */

  if (which_image == SIDE_IMG) {
    xs = (float)(ImgXsize(side_img));
    ys = (float)(ImgYsize(side_img));
  }
  else {
    xs = right_model-left_model-1; /* IMPORTANT!: we assume *_model is a rectangle around
*/
    ys = top_model-bottom_model-1; /* the current projection of the model */
    l  = left_model;
    b  = bottom_model;
  }

  recompute_face();
  prepareMatrix(M);
  for (j=1; j <= np; j++) {
    world2screen(M, left_pt[j][0], left_pt[j][1], left_pt[j][2],
          &left_tex[j][0], &left_tex[j][1]);
    left_tex[j][0] = (left_tex[j][0]-l)/xs;
    left_tex[j][1] = (left_tex[j][1]-b)/ys;
    PRINTF5("%d: l,b=%.1f,%.1f. tex=%f, %f\n",j,l,b,left_tex[j][0],left_tex[j][1]);
    world2screen(M, right_pt[j][0], right_pt[j][1], right_pt[j][2],
          &right_tex[j][0], &right_tex[j][1]);
```

```
        right_tex[j][0] = (right_tex[j][0]-l)/xs;
        right_tex[j][1] = (right_tex[j][1]-b)/ys;
      }
    }
```

CLAIMS

1. A method for automated computerized audio visual dubbing of movies comprising of the steps:

(a) selecting from the movie a frame having a picture, preferably frontal, of the actor's head and, if available, a frame with its side profile;

(b) marking on the face several significant feature points and measuring their locations in the frame;

(c) fitting a generic three-dimensional head model to the actor's two-dimensional head picture by adapting the data of the significant feature points, as measured in stage (b), to their location in the model;

(d) tracking of the said fitted three-dimensional head model parameters throughout the movie, from one frame to its successor, iteratively in an automated computerized way and creating a library of reference similarity frames ;

(e) taking a movie of a dubber wherein the dubber speaks the target text;

(f) repeating stages (a), (b), (c), and (d) with the dubber;

(g) normalizing the dubber's minimum and maximum values of each parameter to the actor's minimum and maximum values for the same parameters;

(h) mapping, on a frame to frame basis, the two-dimensional actors face onto its three-dimensional head model by using a texture mapping technique, making use of reference similarity frames;

(i) changing the texture mapped three-dimensional model obtained in stage (h) by replacing, on a frame to frame basis, the original mouth parameters with the mouth parameters as computed in stage (d) for the dubber and obtaining the parametric description for the new picture, with identical values to the original, except that the actor's mouth status resembles the mouth status of the dubber;

(j) texture mapping the lips area of the same actor from a frame in the movie, with identical or very similar mouth status to the desired new mouth status, onto the lips area of the actor's head

model for the current frame and then projecting the lips area from the actor's head model onto the current new frame (an optional stage).

2. A method according to claim 1 wherein the parameters for controlling the three-dimensional head model are the position, orientation, and expression of the head model mouth.

3. A method according to claim 1 wherein the significant feature points on the face marked on stage b are the eye corners, the mouth corners, and the top and bottom of the face.

4. A method according to claim 1 wherein about 15 significant feature points on the face are used in the tracking stage.

5. A method according to claim 1 wherein the movie to be audio visually dubbed is a sequence of one or more still photographs which are identically duplicated, frame after frame, to create a portion of a movie.

6. A method according to claim 5 for audio visual dubbing of still photographs in TV programs such as news from field correspondents.

7. A method according to claim 5 wherein the actor does not speak such as a baby or a mute person.

8. A method according to claim 1 wherein the original movie is an animated cartoon.

9. A method according to claim 8 wherein the actor is an animal or any other non-human or non-living object.

10. A method according to claim 1 wherein the movies are advertisment movies.

11. A method according to claim 1 wherein the pictures of the movie and of the still photos are in electronic digital form.

12. A method according to claim 1 wherein the pictures are translated into digital form, manipulated in digital form, and returned back to any desired form.

13. A method according to claims 8 wherein the production of an annimated cartoon is assisted by drawing a straight line segment for the actor's mouth, drawing a small actor's picture dictionary

containing representive reference similarity frames with completely drawn mouths, and then allowing these lip-line segments to be replaced with the coresponding lip shapes of the dubber as are to be found in the actor's picture dictionary.

14. A method according to claim 1 wherein the "movie" is a painting, a drawing, or a picture.

15. A method according to claim 1 for creating new movies or new portions of movies from a library of reference frames.

16. A method according to claim 1 for the conversion of background narrative, or any spoken text, to an audio visual form.

17. A method according to claim 1 wherein the dubber speaks the target text in either another language or the same language and the movie of the dubber is taken while the dubber performs a routine dubbing adaptation of the original into target text.

18. A method for creation of a library of reference similarity frames in the method as defined in claim 1.

19. A method according to claim 1 wherein the automated computerised audio visual dubbing of movies is done by the software, as shown in Appendix 1, or by similar software.

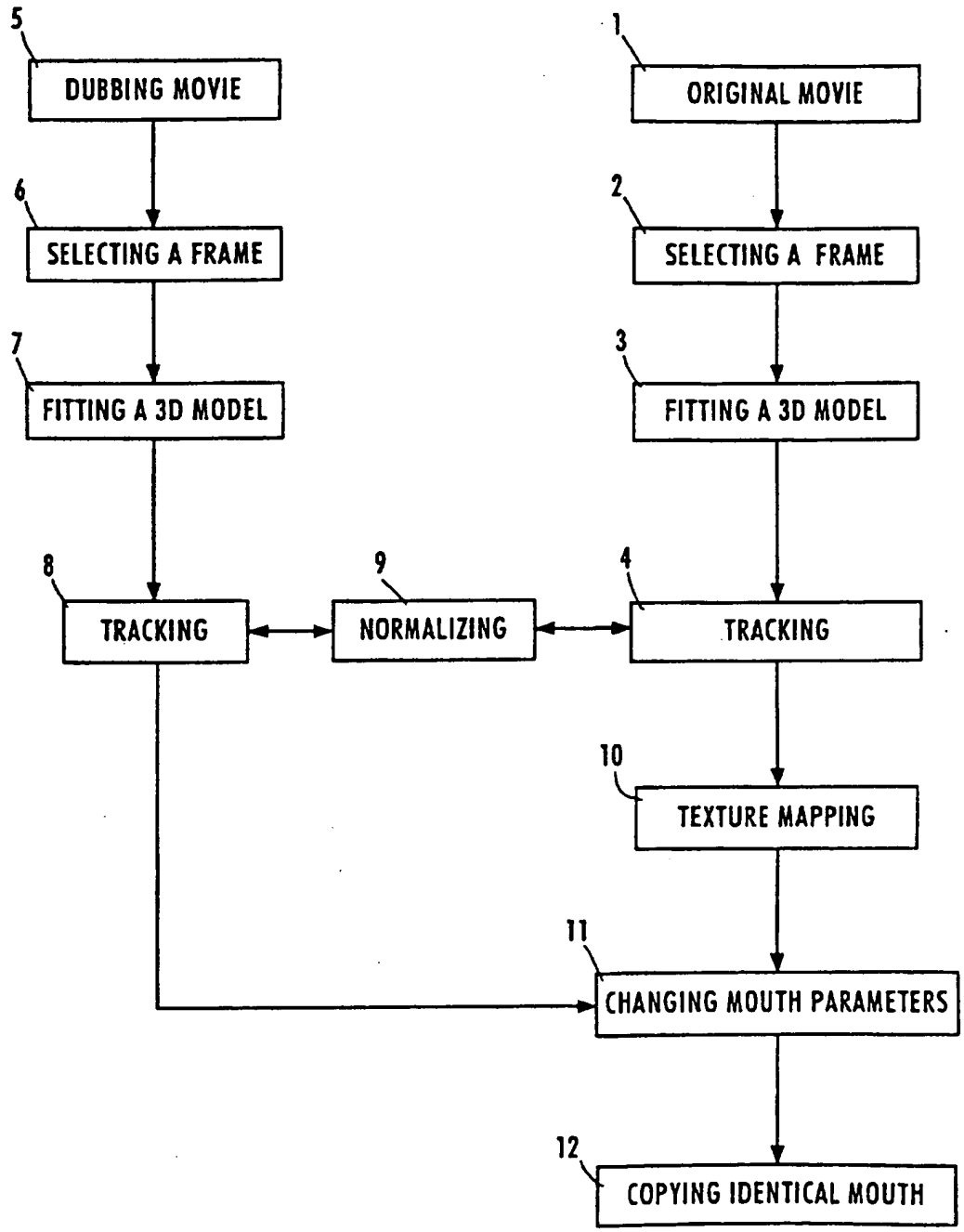
20. A software for use in the method as defined in claim 1.

21. A movie prepared by the automated audio visual dubbing method as defined in the preceeding claims.

22. A movie according to claim 22 wherein the movie is a video movie.

23. A method substantially as herein before described and illustrated.

Figure 1


5
DUBBING MOVIE
1
ORIGINAL MOVIE
6
SELECTING A FRAME
2
SELECTING A FRAME
7
FITTING A 3D MODEL
3
FITTING A 3D MODEL
8
TRACKING
9
NORMALIZING
4
TRACKING
10
TEXTURE MAPPING
11
CHANGING MOUTH PARAMETERS
12
COPYING IDENTICAL MOUTH

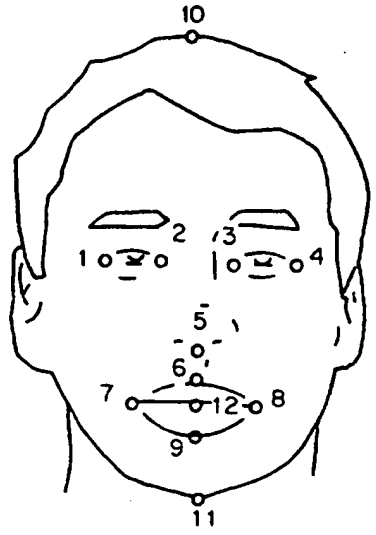


Figure 2a

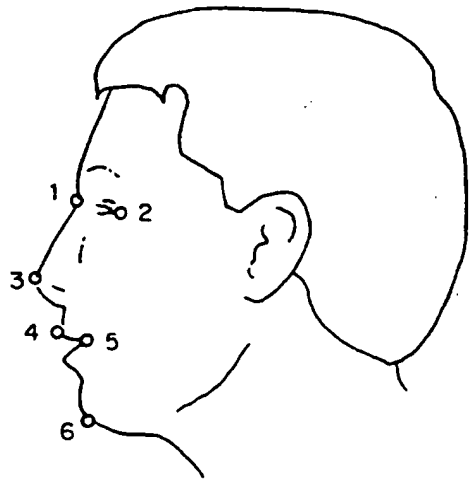


Figure 2b

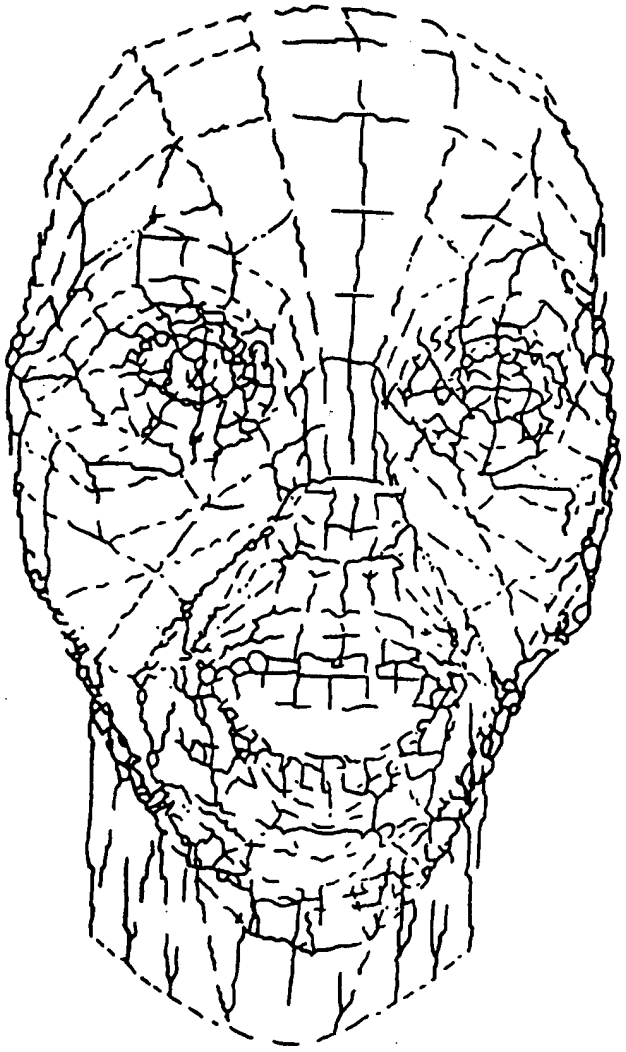


Figure 3

Figure 4a

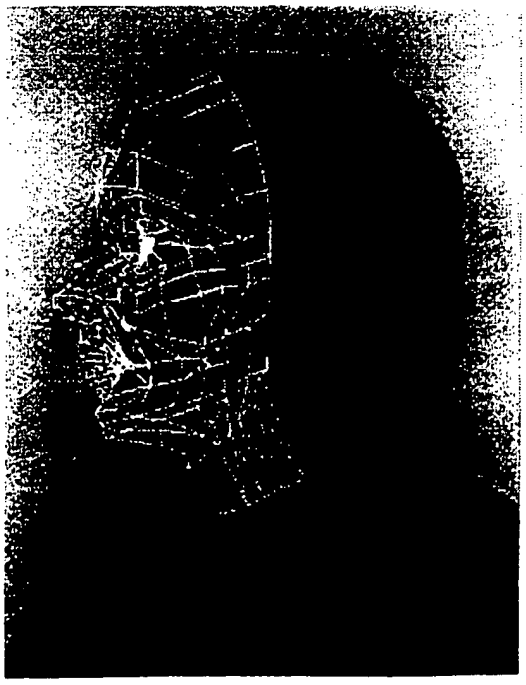


Figure 4b

INTERNATIONAL SEARCH REPORT

International application No.
PCT/IB96/01056

A. CLASSIFICATION OF SUBJECT MATTER
IPC(6) :G11B 31/00; G06F 15/44
US CL :395/173, 327
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 395/173, 327, 806, 807

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

APS
TALKING MOVIES, LIP SYNCHRONIZATION

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
	Please See Continuation of Second Sheet.	

[X] Further documents are listed in the continuation of Box C. [ ] See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search 25 FEBRUARY 1997	Date of mailing of the international search report 26 MAR 1997
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer HEATHER HERNDON Telephone No. (703) 305-9701

Form PCT/ISA/210 (second sheet)(July 1992)*

INTERNATIONAL SEARCH REPORT

International application No.
PCT/IB96/01056

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US, 4,569,026 A (BEST) 04 FEBRUARY 1986 (04/02/86), see entire document, abstract	1-24
Y	MORISHIMA et al. Facial Expression Synthesis Based on Natural Voice for Virtual Face-to-Face Communication with Machine, September 1993, pages 486-489.	1-24
Y	PLATT et al. Animating Facial Expressions. August 1981, pages 247, 249-250.	1-4
Y	NAKAGAWA et al. An Experimental PC-based Media Conversion System. August 1993, pages 141-142.	1-4, 11, 16-24
A	US, 4,884,972 A (GASPER) 05 DECEMBER 1989 (05/12/89), see entire document	1-24
A,P	US, 5,557,724 A (SAMPAT et al.) 17 SEPTEMBER 1996 (17/09/96), see entire document	1, 7-24